PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 5 : G06F 15/403

A1

(11) International Publication Number: WO 94/20915

(43) International Publication Date: 15 September 1994 (15.09.94)

(21) International Application Number: PCT/CA94/00111

(22) International Filing Date: 1 March 1994 (01.03.94)

(30) Priority Data:
2,090,852 2 March 1993 (02.03.93) CA

(71) Applicant (*for all designated States except US*): 1008495 ONTARIO LIMITED [CA/CA]; 57 Center Street, Thornhill, Ontario L4J 1G2 (CA).

(72) Inventor; and

(75) Inventor/Applicant (*for US only*): IOSSIPHIDIS, Joseph [CA/CA]; 28 Woodmans Chart, Markham, Ontario L3R 6K8 (CA).

(74) Agents: HUGHES, Ivor, M. et al.; 175 Commerce Valley Drive West, Suite 200, Thornhill, Ontario L3T 7P6 (CA).

(81) Designated States: AT, AU, BB, BG, BR, BY, CA, CH, CN, CZ, DE, DK, ES, FI, GB, HU, JP, KP, KR, KZ, LK, LU, LV, MG, MN, MW, NL, NO, NZ, PL, PT, RO, RU, SD, SE, SI, SK, UA, US, UZ, VN, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

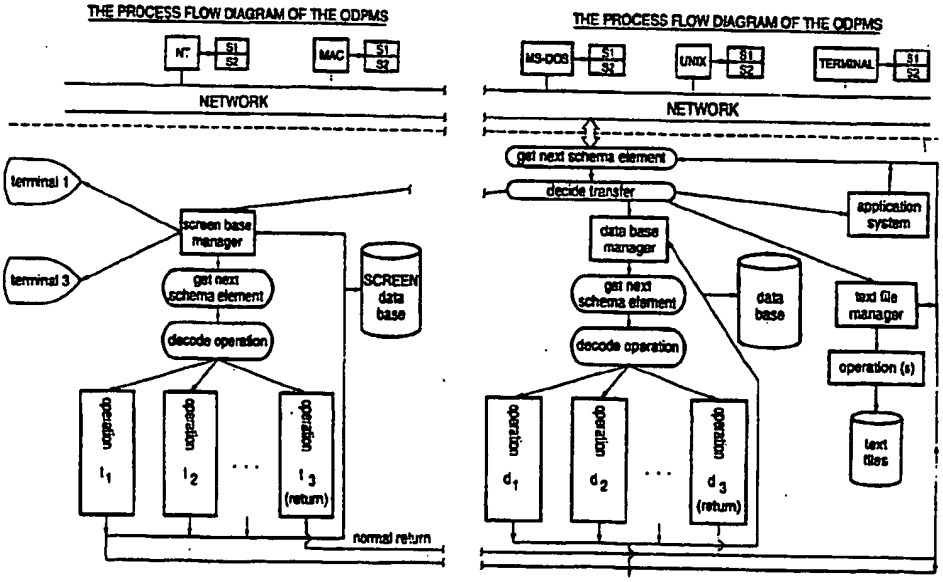
Published

With international search report.

Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: GENERIC DATA AND PROCESS BASE MANAGER AND DATA AND PROCESS BASE MANAGEMENT SYSTEM FOR HETEROGENOUS DATABASES AND PROCESS BASES

(57) Abstract



A procedural relation processing system is fundamental in managing distributed heterogeneous databases and distributed heterogeneous process bases, or application systems, at run-time, as they are perceived by the users, directly at the time and place of perception. A procedural relation determines, as a unit, a variable combinational process and a variable structure operand, to which the combinational process is applied. A structure operand can be any group of data existing in a database or any group of programming statements existing in a subroutine, or procedure, of an application system. A procedural relation processing system is fundamental in distributed heterogeneous databases and distributed heterogeneous process bases since it can represent and process any data or process structure. A user can express a procedural relation on the terminal or in an application program, as it is perceived by the user at run-time, directly at the time and place of perception. A procedural relation processing system, consisting of an Open Data and Process Model (ODPM) and associated management system components will be referred to as an Open Data and Process Base Management System (ODPMS). The ODPMS, accepts as input, the procedural relation. After analyzing its syntax and semantics, the ODPMS transfers control to the location on the network where the data exists, identifying to the remote application system what statements to execute. These statements issue a request to their local Data Base Management System (DBMS). After receiving the required data values, the remote application system returns the data values to our local area and these values are displayed on the terminal by the ODPMS.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	GB	United Kingdom	MR	Mauritania
AU	Australia	GE	Georgia	MW	Malawi
BB	Barbados	GN	Guinea	NE	Niger
BE	Belgium	GR	Greece	NL	Netherlands
BF	Burkina Faso	HU	Hungary	NO	Norway
BG	Bulgaria	IE	Ireland	NZ	New Zealand
BJ	Benin	IT	Italy	PL	Poland
BR	Brazil	JP	Japan	PT	Portugal
BY	Belarus	KE	Kenya	RO	Romania
CA	Canada	KG	Kyrgystan	RU	Russian Federation
CF	Central African Republic	KP	Democratic People's Republic of Korea	SD	Sudan
CG	Congo			SE	Sweden
CH	Switzerland	KR	Republic of Korea	SI	Slovenia
CI	Côte d'Ivoire	KZ	Kazakhstan	SK	Slovakia
CM	Cameroon	LI	Liechtenstein	SN	Senegal
CN	China	LK	Sri Lanka	TD	Chad
CS	Czechoslovakia	LU	Luxembourg	TG	Togo
CZ	Czech Republic	LV	Latvia	TJ	Tajikistan
DE	Germany	MC	Monaco	TT	Trinidad and Tobago
DK	Denmark	MD	Republic of Moldova	UA	Ukraine
ES	Spain	MG	Madagascar	US	United States of America
FI	Finland	ML	Mali	UZ	Uzbekistan
FR	France	MN	Mongolia	VN	Viet Nam
GA	Gabon				

TITLE OF INVENTION

GENERIC DATA AND PROCESS BASE MANAGER AND DATA AND PROCESS BASE MANAGEMENT SYSTEM FOR HETEROGENEOUS DATABASES AND PROCESS BASES

FIELD OF INVENTION

The present invention relates to a procedural relation processing system and more specifically to such a system necessary for managing heterogeneous distributed databases and/or heterogeneous distributed process bases.

BACKGROND OF INVENTION

Current database management techniques cannot be used directly to manage heterogeneous database systems. To accomplish this, independent programs must be written to translate logical data views between heterogeneous databases. (A logical data view; is an independent combination of data determined or expressed by a user who may be a programmer or an end-user).

The current difficulty in managing heterogeneous databases is due to the way DBNSs are defined and function. Under any DBMS the data management processes are dependent on the data structures. Current database management techniques cannot be used directly to manage heterogeneous databases. To accomplish this, programs must be written to translate logical data views between heterogeneous databases. A logical data view, is an open combination of data determined by a programmer. Formally, this may be expressed in the following functional relationship:

$$f: D \rightarrow P$$

where D represents a logical data structure, and

P represents a set of management processes.

This implies that when we express a data association between two records which exist in two different files, the same management processes are executed at all times under the same DBMS. The problem arises because that these data management processes are not the same under a relational, hierarchical or network DBMS.

A Distributed Data Base Management System (DDBMS) is a set of local Data Base Management Systems (DBMS) in different physical locations connected to each other by various networks. Each DBMS has a computer and memories, wherein the databases are stored. A database is a set of related data files. The rules as to how one can relate or combine data that belong to the same data file or to different data files of a database

are determined by the database specific data model. Thus, the role of a data model, is how to abstract the structures and the structural operations of a database. DBNSs that use the same data models, are referred to as homogeneous and DBNSs that use different data models, are referred to as heterogeneous.

Heterogeneous DMBSs, and sometimes even homogeneous DBMSs cannot communicate with each other. The problem resides mainly with the specific data models of the databases. Different data models describe different structural operations on the data of a database. For heterogeneous databases to communicate, a database translation between the source and the target DBMSs is needed.

There are two general solutions to this problem:

(1) an interpretive translator, (see Figure 1), and

(2) a translator generator, (see Figure 2).

However, data models, DBMSs, database translators, etc., are computer oriented because they incorporate only computer constructs; human interaction is absent from current DBMSs.

DDBMS are complex systems because they have many static and dynamic components: DBMSs, data models, schemas, DDLs (Data Definition Language), DMLs (Data Manipulation Language), databases, database translation, subschemas, etc. Although there are many different DBMSs, they introduce the same problem.

(1) Data models cannot be used as communication mediums.

The data models are computer oriented and not human oriented. That is, they should allow users to combine, or relate, data as they are perceived by the users, right at the time and place of perception. The data models, as known, represent abstract combinations of data in tables (relational data models), or graphs (hierarchical, DBTG (Data Base Task Group), network data models). The natural representation of communication between human and computers would be dialogue or text in natural language.

(2) Data models are data oriented and not information oriented.

A data model represents allowable combinations of data. However, information is more than a combination of data. A DBMS manages data existing in data files. An information management system is defined in the Oxford Science Publications Dictionary of Computing. 1986, on page 182:

> "A term not yet in common use, and that may best be understood by contrast with data management system. A data management system is one that, because it operates on data, has relatively well-defined syntax, semantics, and operations. The contents of an information management system include other forms of information (e.g. text); its syntax and semantics are accordingly less capable of precise definition, and its effective use is much more dependent on interaction with users."

(3) Data models have inherent constraints.

Constraints are restrictions on the combination of the data in a database (for example hierarchical, relational, or network). This means that the data in a database cannot be related in any conceivable combination. The structures of a data model, disallow certain relations. For example, in a hierarchical data model, a child record type cannot have more than one parent record type. By definition the constraints are determined by the structures of a data model. This statement implies that the structures of a data model determine the semantics on the combinations of the data. In other words, a data model generalizes the data management process by allowing the structures to determine these data management processes. Thus, in a DBMS the variable components are the data combinations, where data management processes are fixed. That is, for a given combination of data, the same data management processes are executed. This is the main reason that heterogeneous DBMS cannot communicate: the data models generalize the data management process.

(4) DBMSs offer fixed logical data views.

A DBMS expects a user to express, in advance, the required logical data view, or the data that a user requests. This logical data view is considered by a DBMS as a unit. That is, once a logical data view has been defined, it becomes fixed. This restriction implies three main things:

(a) a DBMS is a closed system and cannot interact with the

external world, (humans and application programs),

(b) a user must foresee, in advance, all the data required for a decision, and

(c) a user must be able to assign other semantics to certain data values.

For example, let's assume that a DBMS monitors the temperature of a machine. Different temperatures of the machine imply different semantics or actions to be taken by an operator. There are two solutions to this problem. Either the operator knows the semantics himself or the DBMS is able to assign the semantics to the operator, as they occur.

(5) DBMS are mostly designed for business applications.

DBMS have been defined for business applications. However, DBMS can be used for scientific applications, image processing applications, etc. Traditionally DBMS claim tacitly, that their relations are the totality of relations that a user may require for a real life problem. Of course, in reality this is not the case.

(6) Computer Specialists generate the logical data views.

For users to be able to generate a logical data view under current DBMS they must write an application program as it has been defined by the Data Manipulation Language (DML) and as the structures have been defined in the schema. Computer specialists have this knowledge, not the managers or executives who need the data. It is obvious that the specialists must know what type of information is being used and how, which could be sensitive.

Furthermore, changes in information needs of the managers and executives must wait for the specialist to implement them. It is assumed then that programmers know what and how information the managers and executives must extract, and perhaps the reasons why. Moreover, programmers determine what information will be available to the managers and executives. This is not acceptable.

(7) Who interacts with the DBMS.

Under current DBMS, an application program is the only external component that can interact with the DBMS. As described above, this interaction is not dynamic but static.

The Applicant is aware of the following prior art which exemplify the problems described above. In providing the prior art Applicant makes no admissions as to the relevance of any portions of the

patents or the entire documents.

United States Patent No. 5,129,086, assigned to International Business Machines Corporation, refers to a System and Method for Intercommunicating Between Applications and a Database Manager.

United States Patent No. 5,058,000, assigned to Prime Computer, Inc., refers to a System for Accessing Remote Heterogeneous Databases including Formatting Retrieved Data into Applications Program Format.

United States Patent No. 4,894,771, assigned to Ricoh Company, Ltd., refers to a Data Base Management System Extending Structure.

United States Patent No. 5,097,533, assigned to International Business Machines Corporation, refers to a System and Method for Interfacing Computer Application Programs Written in Different Languages to a Software System.

United States Patent No. 4,791,561, assigned to Wang Laboratories, Inc., refers to an Interactive Construction of Means for Database Maintenance.

United States Patent No. 4,774,661, assigned to American Telephone and Telegraph Company, refers to a Database Management System with Active Data Dictionary, wherein said dictionary is used to generate and manipulate subschema for an application process interface. The system is not open but fixed once defined, unlike Applicant's structure described below.

Nowhere within the prior art is there found a database and process base management system for distributed heterogeneous databases and process bases.

It is therefore a primary object of the present invention to provide a database and process base management system for distributed heterogeneous databases and process bases.

It is also a primary object of the present invention to allow users to combine data as they are perceived by the users, right at the time and place of perception. The dialogue between humans and computers is done in a symbolic form, because symbols can be processed faster, they are shorter than words, they are less ambiguous; and they can be universal.

It is therefore another object of the present invention according to the above definitions unlike the prior art to provide a super

information management system that manages, simultaneously, at run-time;

(a) databases,

(b) displayed text on the terminals,

(c) text files, and

(d) process bases.

It is therefore yet another object of the present invention to resolve the problems of the prior art DBMS by introducing the Open Data and Process Model (ODPM), the data combinations and the data management processes being both variable and determined by the users.

It is still yet another object of the present invention to allow users to determine new logical data views as they are required at the time and place they are perceived, enabling users to not only access a subset of the database allocate to them, but the entire database, their entire process base, and their text files.

It is a primary object of the present invention to provide an Open Data and Process Model (ODPM) that assumes by definition that it cannot automatically represent, any conceivable relation expressed by a user but it can represent any relation that a user may request at run-time, simply by accepting and interpreting semantics of unspecified relations at run-time. In effect it renders said ODPM as a double DMBS which firstly manages data as described in the schema, and secondly, as a DBMS that manages data by accepting run-time data management symbols.

It is therefore yet another object of the present invention to allow managers and executives to extract the information they require, as they require it and displays the information at run-time, when it is asked, including:

(a) what relations a user can express from a given point,

(b) the explanation of the semantics of the symbols being used,

(c) the entire description of the application system,

(d) etc. (see pages 39 for a description of the run-time symbols).

Thus, under the objects of the present invention, an end-user may be defined as the run-time programmer that does not have to write application programs. The external components that can interact with the present invention are as follows:

(a) end-users,

(b) application systems, and

(c) schemas.

The interactions take place at run-time and are dynamic. That is, they impose demands to change the state of a database, a process base, or a screen base.

Further and other objects of this invention will become apparent to a man skilled in the art when considering the following summary of the invention and the more detailed description of the preferred embodiments illustrated herein.

SUMMARY OF THE INVENTION

The present invention overcomes the problems of the prior art by eliminating the data management dependency on the data structure. This is accomplished by the introduction of the procedural relations.

A procedural relation is a symbolic string expressed in the schema or at run-time by users. It represents a data management process and a data structure variable, to which the data management process is applied, as a unit.

Under this scheme, any data structure can be interpreted whether it be relational, hierarchical, network, or any other structure. Formally, the functionality of an Open Data and Process Model (ODPM) may be expressed in the following functional relationship:

$$f: (P,D) \longrightarrow \{^{D}_{Pa}$$

where D represents a data structure variable,

P represents a data management process(es), and

Pa represents an application process.

The present invention provides a multi-user database and application management system based on symbolic procedural relations expressed in the schema or at run-time by end-users and/or programmers.

A procedural relation processing system, consisting of an Open Data and Process Model (ODPM) and associated management system components will be referred to as an Open Data and Process Base Management System (ODPMS).

The ODPMS is designed to allow a user to access data in a manner which the user determines directly at run-time. It allows for the logical manipulation of data and application processes in a dynamic and random manner directly at the time of use.

The ODPMS was developed as a computer system that can interface with any computer language in existence that uses a CALL statement, such as C, COBOL, FORTRAN, PASCAL, ETC. Not only can the ODPMS communicate with other DMBSs (Data Base Management Systems), but it can do so without the need for any independent translator. Therefore, its reaction time is faster than any potential solution to this problem which companies are attempting to develop using independent translators. The ODPMS allows end users to access information at run-time without needing to know by what method they are actually retrieving the data. In other words, no traditional programming or particular expertise is required by the end user.

An example of a practical application the ODPM may be found where an end user at one branch of a large network would want to access and retrieve information from another branch (the target branch) which has adopted a different data storage and retrieval system. This would normally require a certain type of translation software to be in place, on a case to case basis. The end user would also need to possess a certain amount of technical and practical knowledge on the system in use at the target branch. The ODPMS would solve this problem by allowing the end user to access and retrieve information from the target branch with no further knowledge than that required to access and retrieve information at his own branch. Effectively, the ODPMS is a system by which compatibility between major computer networks and DBMSs around the world may be achieved. Another feature of this system is that it is able to internally build a whole new level of security that to this point has not been available.

The ODPM does not have any inherent data structure or process structure. It is totally open to the user.

The ODPMS is a run-time, multi-user, data and process structure management system based on schema information and run-time input specified by the users. It functions as:

(a) a user interface, reading and executing commands from the terminals;

(b) an interpreter, reading and executing command from a program;

(c) a DBMS, offering a model for data and processes, along with query and update facilities; and

(d) a data server, managing data requests and transfers among processes.

The ODPMS simultaneously manages the schema, the programs and all of its end-users (the users) allocating executions control as required. Every user of the ODPMS functions in a client/server mode. For example, when the ODPMS transfers execution control to an end-user, that user can express their own commands which can take priority over schema or program commands.

As a DBMS, the ODPMS has its own data model called the Open Data and Process Model (ODPM). Under ODPM, the smallest manageable unit of data is the data segment which is a non-empty set of data in a one-to-one relationship. A data segment can be just one data element, or a subset of a record or the record itself. Although the definition of a data segment is fixed, it appears open to the user. That is, a data segment in reality is a variable set of data elements. This variability can be expressed by any user with the only restriction being their receiving permission to do so.

The ODPM has may types of data segments. If a data segment is type 'program' then this data segment uniquely identifies a set of statements of a procedure called a program segment. A program segment, like a data segment, is a set of programming statements, or a subset of a procedure, or the procedure itself. The statements of a program are open to the programmer.

The programmer or the Data and Process Base Administrator (DPBA) determines the language and the semantics assigned to the program segment which can be written in any language that has any CALL statement. Although a programmer writes the statement of a program segment, their management is determined by the ODPMS. Thus, when execution control is transferred to a procedure, the procedure must first interpret which program segments to execute. After executing the program segment, the procedure must call the ODPMS.

The relational operators of the ODPM can be:

one-to-one;

one-to many;

many-to-many;

circular or loops, and

recursive

However, what makes the ODPM different from any other DBMS is the semantics assigned to the relations. These semantics are expressed in BNF (Backus-Naur Form) notation. The major characteristics of the relations are:

(a) the management processes may be undetermined;

(b) the related object may be undetermined;

(c) the relations have the highest priority; and

(d) the objects of the relations are interruptible.

With a relation or an association in DBMS, the related 'thing' must be another data record and thus, the nature of the related 'thing' is fixed. Under the ODPM, a user determines what can be related. It could be:

a data element;

a data segment;

a data record;

a function;

a list or a menu;

plain English instructions, or

the syntax and semantics of the ODPMS's open operators.

As soon as a relation is introduced by a user, the ODPMS processes it immediately. For example, let us say the process of a screen record is as below.

THIS FUNCTION ADDS PROPERTIES

(1) PROP ID : 1

(2) ADDRESS1: 123 Bloor St.

(3) ADDRESS2:

At the third attribute (i.e. ADDRESS2), the user wishes to see some statistical results defined on the screen record 'STATISTICS'. The user has merely to type in: <'STATISTICS'. The ODPMS interrupts the process of the screen record 'ADD PROPERTIES' and transfers execution control to the screen record 'STATISTICS'. When the process of that screen record is finished execution, control goes back to screen record' ADD PROPERTIES' at the previous position of the third screen attribute or element. However, before the process of the screen record 'STATISTICS' was finished, a user could have associated another screen record, and so on. Thus, we could form a chain of screen records in different levels of execution.

Unlike a DBMS which allows a user to express one and only one subschema at a time the ODPMS allows a user to generate a set of subschemas at run-time. This makes the whole database and the whole process base available to the user if authorized, whereas in DBMS, a subschema is an indivisible data view. Even a large number of types of relations as exist in an object oriented database may not be sufficient to make the interface of users and a database as flexible as the ODPMS.

The ODPMS makes the environment of a database friendly by using 'open operators' which can be expressed by any user at run-time. An end-user uses run-time symbols to express an open-operator anywhere in the physical screen that the ODPMS expects input. A programmer uses the Program Status Vector (PSV) to express open-operators. The PSV appears in the argument list of a CALL statement to the ODPMS. The CALL to the ODPMS is universal in syntax, and is independent of the function of the procedure or the semantics assigned to open-operators.

An open-operator is in reality the set of all the run-time symbols expressed by users. Thus, the relations are a subset of the open-operators. An open-operator can express:

informative demands;

process demands, and

relational demands

An informative demand can inform the end-user of the type of the expected input (i.e. character, integer,..) and the number of characters. Another informative demand can inform the end-user as to what record types can be related from a particular screen element. Thus, an end-user can determine the entire portion of the database assigned to them and what data they can add, display, modify, or delete without writing a program or knowing the schema structures.

Process demands allow users to control the process on a screen record. For example, an end-user can stop the process of a screen record, or can ignore the entire entered input.

A relational demand is simply a procedural relationship.

As a database and process base management system the ODPMS allows any data record of a database and any procedure of a process base to be shareable among many users with diverse structural or relational needs. The DPBA controls what open-operators are assigned to each user for every screen record. There can be end-users with no process

or relational operators and in this case, they are driven by the schema. This may mean that the process and the relational operators can occur at fixed places of a screen record, as opposed to open or random occurrence.

An application system as viewed by the ODPMS is a set of related procedures which are determined in the schema but there may be procedures that do not interact with the ODPMS at all. However, a programmer does not have to know what procedures are related or what they do because a procedure is a set of program segments related with open or non-deterministic relations.

As stated earlier, a procedure may be written in any language that has a CALL statement but the semantics of the procedure and consequently, the semantics of the program segments are openly determined by the DPBA. However, the results and demands of a program segment are known by the ODPMS. That is to say, a procedure cannot destabilize the state of the ODPMS. A procedure may be used to carry out complex arithmetic operations, or to carry out complex conditional operations, etc.

A screen database is a set of related screen records. These relationships are determined in the schema or expressed by users at run-time. A user is not concerned with the functionality of a screen record or how it is related to anything else. A screen record is a set of variable data segments related with open or non-deterministic relations. The DPBA can determine to where a data segment is mapped. For example, a data segment may be mapped into a data file of a database or to a program segment of a procedure, or both. In reality, the ODPMS is instructed as to who is the producer and who are the consumers, who utilize the generated data values.

The ODPMS schema is more than a schema as we know it. Generally speaking, a schema represents a database in micrography or it represents a pseudo database. That is, it determines how to map logical views into the database. However the ODPMS schema determines how a schema is to be manipulated; under the ODPMS, a schema is a variable entity, not a fixed one.

A database is a set of related data files. A data file is a set of data records, which is a set of data segments. How the data files, the data records, and the data segments are related is determined in the schema. A data file can be accessed by a key or keys, or by non-key data values, or by

linkages. The ODPMS uses a grammar to express its functionality.

All the compilation phases of the ODPMS are table driven and thus, modification is easy. As well, it uses dynamic storage allocation to build all the data buffers and data compaction for the structures and schema. An application system does not have to be recompiled every time it uses the ODPMS as it is compiled once over its entire life.

An ODPM defines a set of structure variables existing in different hardware parts of a computer system that can be combined with open structural interpretations, or open structural semantics. The structure variables can exist in:

(1) screen databases,

(2) databases,

(3) text bases, and

(4) process bases.

(1) Definition and structures of screen databases.

A screen database is a set of screen records stored in data files. A screen record contains fixed text whose contents are determined by the DPBA. A screen record may be defined as the entire fixed text displayed on the physical screen of a terminal. A screen record is associated with a model record. (see definition of model record in Figure 5).

(2) Definition and structures of databases.

A database is a set of related data files. A data file is a set of homogeneous data records. A data record is a set of heterogeneous data values. A relation between two data files is determined by a physical procedural relation. A physical procedural relation determines two things simultaneously:

(a) the relational structure, and

(b) the relational (or the combinational) interpretation(s).

The relational structure identifies the records of the related files, and the combinational interpretation determines how to interpret the relational structure. In the case of the relational DBMS, the relational structure is empty. A relational structure itself is stored in data files.

(3) Text Bases

A text base is a set of text records stored in data files. A text record contains information about something, (i.e. the documentation of an application program, etc.). A text record is identified by a unique name

assigned by the DPBA. A text record may require several physical screens to be displayed. A user determines when the next physical screen is to be displayed or when the text process is to be stopped.

(4) Process Bases

A process base is in reality, an application system. An application system is a set of related subroutines, or procedures. Each subroutine is uniquely identified by a name. This name has been determined by the DPBA. A subroutine is divided into a set of program segments. A program segment is a set of programming statements, identified as a unit, and written in any computer language that has a CALL statement. The semantics of a program segment is open to the DPBA. A subroutine can be shareable among many users, simultaneously, with different combinations of program segments.

The OPDM

The ODPM distinctly identifies the structure variables of the model and the structural interpretation between the structures. The largest structural unit is a model record. A model record is uniquely recognized by a name defined by the DPBA. The relation between two related model records can be fixed or open. When a relation is fixed, we know:

(a) which is the related model record, and

(b) what structural interpretation to assign.

When a relation is open we know what is the domain of the related model records but, we do not know what structural interpretation to assign. This is known after the related model record is identified.

A model record is further divided as a set of disjointed model segments. A model segment is uniquely recognized within a model record by a name defined by the DPBA. A model segment can be related with other model segments that belong in the same model record or with other model records, by fixed or open relations. The same semantics are applied here as before.

A model segment represents the creation and distribution of data values in the entire network and can be defined by the tuple:

<data, creation of data, structural interpretation, distribution of data, procedural relation or relations>

The 'data' determine the data types of the values. For example, integer, real, character, etc. The creation of data determines who generates the data

values for this model segment. It can be an end-user, a subroutine or subroutines in the network, or a DBMS in the network. The structural interpretation determines what structural semantics to assign, to identify the data if they are extracted from a database. The distribution of data determines the consumer(s) of the information in the network. The consumers can be end-users, a subroutine, a program segment, or a DBMS, or any combinations of the above.

The procedural relation determines what procedural relations an end-user or a subroutine can express at any place of a model segment. We can explicitly determine the procedural relations assigned to the users. Thus, users with less computer experience may be assigned none or very few procedural relation capabilities until they gain more experience. The DPBA can change this capability at any time. Users with computer knowledge may be assigned the entire procedural relation capability or subsets of it. One must understand that the procedural relation capability is explicitly determined at the model segment levels. That is, the procedural relation capability for the same user can vary.

It is apparent from this description that an ODPM does not identify relations on data, like conventional data models, but procedural relations on data. The dynamics of a procedural relation is described below.

THE PROPERTIES OF PROCEDURAL RELATIONS

A procedural relation determines, (a) a symbolic combinational (or data management) process required to carry out the relation, and (b) a symbolic structure (data or process) variable that is related. A combinational process symbol is a universal symbol. That is, the same combinational symbol is used for the same type of structure variables in the entire system.

The symbol required to determine the combinational process is called a combinational operator and the symbol required to determine the related structure variable is called a combinational operand.

The ODPMS recognizes and processes combinational operators and combinational operands expressed dynamically at run-time. The ODPMS reduces any data structure and any application program structure to a few combinational operators and a few combinational operands.

A procedural relation in BNF has:

<procedural relation> :: = <combinational operator> <combinational operand> wherein:

< combinational operator > :: = 'c_1 ' | 'c_2' | . . . | 'c_n'

< combinational operand > :: = 'd_1 ' | 'd_2' | . . . | 'd_n' | 'p_1' | 'p_2' | . . . | 'p_n'

wherein $c_1, c_2, \ldots c_n$ represent symbols with preassigned combinational semantics;
and wherein $d_1, d_2, \ldots d_n, p_1, p_2, \ldots p_n$ represent symbols that represent types of data structure variables and types of application program structure variables.

It is apparent that the OPDMS makes the management of data and application processes look like expressions of symbols determined by users. There are more semantic rules on the procedural relations that are not apparent by the formal descriptions above. They are:

(1) Open Structural Semantics

A user determines how a relation between two logical views (as it is setup in schemas in data structure variables) is to be interpreted. That interpretation may use relational, hierarchical, network, or other structural rules.

(2) General Purpose

A procedural relation can combine, not just data values as do conventional DBMS, but also, application processes and text data. That is, the related thing is not fixed, but variable.

(3) Position Independent

A procedural relation can be expressed unconditionally at any point of a screen record where a user can insert input. There are no restrictions other than the ones determined in the schema by the DPBA.

(4) Direct

Once a procedural relation is expressed by a user, this new procedural relation is executed immediately by the ODPMS. A new procedural relation has the highest priority over any other process. Thus, before a procedural relation has been completed, a user can express a new one.

(5) Interruptible

A procedural relation can be interrupted before the process in its combinational operand has been completed. This is apparent from characteristic (4) above. Thus, before a procedural relation has been completed, a user can express a new one.

(6) Nested

It is apparent from characteristics (4) and (5) that a user can express a sequence of nested procedural relations in any combination.

It is also apparent that a procedural relation system creates dynamic and variable length subschemas providing logical data views, and application programs that exist in many geographical areas.

The ODPMS dynamically manages heterogeneous distributed databases and heterogeneous distributed application systems with open structural semantics wherein:

- a database is a set of related data files.
- a process base is a set of related subroutines or procedures.
- heterogeneous databases are a set of databases distributed over many geographical areas with different data structure rules.
- heterogeneous application systems are a set of application systems (each one written in a different 3GL, Assembly, or any language that has a CALL statement) with different process structure rules.
- open structural semantics represent the management interpretation of a relation.

As described in the background of the invention the problem is that known data management processes are not the same under a relational, hierarchical, or network DBMS. The ODPMS overcomes this problem in eliminating the data management process dependency on the data relations. This is accomplished by the introduction of the procedural relation.

A procedural relation is a symbolic string expressed in the schema or at run-time by users. It represents as a unit, a data management process and a data structure variable, to which the data management process is applied. Under this scheme, we can interpret any data structure whether it be relational, hierarchical, network, or any other data structure. Formally this may be expressed in the following functional relationship:

$$f: (P,D) \longrightarrow \{^{D}_{Pa}$$

where D represents a data structure variable,

P represents a data management process, and

Pa represents an application process.

THE AXIOMS OF THE ODPMS

Many interpretations of what is the ODPMS, could exist. In our case, it is sufficient to say that:

The ODPMS manages procedural relations.

A procedural relation determines, (a) a symbolic combinational (or data management) process required to carry out the relation, and (b) a symbolic structure (data or process) variable that is related. A combination process symbol is a universal symbol. This is, the same combinational symbol is used for the same type of structure variables in the entire system.

The symbol required to determine the combinational process is called a combinational operator and the symbol required to determine the related structure variable is called a combinational operand. Thus, another interpretation of the ODPMS is:

The ODPMS recognizes and processes combinational operators and combinational operands expressed dynamically at run-time.

The ODPMS reduces any data structure and any application program structure to a few combinational operators and a few combinational operands.

Other interpretations for the ODPMS are:

The ODPMS allows users to define their own relations and the structural interpretations, directly at the moment and place of perception.

The ODPMS is used for creating dynamic and variable length subschemas and application programs that exist in many geographical areas.

To these ends and objectives therefore according to one aspect of the invention there is provided an Open Data and Process Base Management System (ODPMS) for a computer installed network of heterogeneous distributed databases and process bases resident on computing means comprising

- means for preparing a schema by the Database Process Base

Administrator (DBPA) for preferably each local ODPMS so as to define a screen database manager, a database manager, a text base manager, and a process base manager, and the manner in which control may be moved to and from subschema elements by the ODPMS

- means for providing procedural relations allowing a user to access and associate data or processes or text as desired from one or more databases or process bases at run time via combinational operators and operands, preferably as per the following relationship:

<procedural relation> :: = <combinational operator> <combinational operand>

wherein:

< combinational operator > :: = 'c_1' | 'c_2' | . . . | 'c_n'

< combinational operand > :: = 'd_1' | 'd_2' | . . . | 'd_n' | 'p_1' | 'p_2' | . . . | 'p_n'

wherein $c_1, c_2, \ldots c_n$ represent symbols with preassigned combinational semantics;

and wherein $d_1, d_2, \ldots d_n, p_1, p_2, \ldots p_n$ represent symbols that represent types of data structure variables and types of application program structure variables,

wherein said user may freely associate any information, text or process at run time via said procedural relations;

- means for circumventing the data model or schema of the heterogeneous distributed database or process base to therefore provide data, or processes responsive to a call statement, as desired by the user under control of the ODPMS

- whereby the user may, if authorized to do so in the schema, logically variably associate data and or text and execute processes on data, from databases and process bases with dissimilar data models, semantics, and protocol on a open variable user defined query basis.

According to yet another aspect of the invention there is provided means for accessing data resident on at least one computer in at least one database and preferably processes in at least one process base, of a known structure (for example hierarchical, network, or relational) comprising:

- schema means defined by a Database and Process base Administrator (DBPA) for providing a database, process base, text base and screen base,

- means for associating information in a manner outside of the structure and protocol or rules of said database on a logical variable open basis without the need for a specific Application Program Interface,
- means for enabling the user to make user defined open associations at run time without the need of a programmer by well defined procedural relations including combinational operators and combinational operands defined by the following definitions:

<procedural relation> :: = <combinational operator> <combinational operand>

wherein:

< combinational operator > :: = 'c_1' | 'c_2' | . . . | 'c_n'

< combinational operand > :: = 'd_1' | 'd_2' | . . . | 'd_n' | 'p_1' | 'p_2' | . . . | 'p_n'

wherein $c_1, c_2, \ldots c_n$ represent symbols with preassigned combinational semantics;

and wherein $d_1, d_2, \ldots d_n, p_1, p_2, \ldots p_n$ represent symbols that represent types of data structure variables and types of application program structure variables,

wherein said user may freely associate any information, text or process at run time via said procedural relations.

In a preferred embodiment of the invention there is provided a procedural relation processing system for managing a network of distributed heterogeneous databases and distributed heterogeneous process bases , or application systems resident on storage media of at least two distinct computers on the network, at run-time, as they are perceived by the users, directly at the time and place of perception, said procedural relation processing system may further comprise procedural relations which determine, as a unit, a variable combinational process and a variable structure operand, to which the combinational process is applied, a structure operand being a group of data existing in a database or a group of programming statements existing in a subroutine, or procedure, of an application system, said procedural relation processing system for representing and processing any data or process structure which a user can express as a procedural relation on a terminal of a computer or in an application program, as it is perceived at run-time, directly at the time and place of perception, the procedural relation processing system accepting as

input, the procedural relation, wherein said management system after analyzing the semantics of the procedural relation, transfers control to the location on the network where the data exists, identifying the remote application system and what statements to execute, the statements issuing a request to the local Database Management System (DBMS) where the data exists which after receiving the required data values, returns the data values to the user's local area computer and these values are displayed on the terminal by the system.

According to yet another aspect of the invention there is provided an open data and process base management system (ODPMS) preferably for use with a network of heterogeneous distributed databases and process bases, said ODPMS comprising data structures and process structures which logically and variably associate data and processes according to a procedural relation as defined by the following functional relationship:

$$f: (P,D) \longrightarrow \{^{D}_{Pa}$$

where D represents a data structure variable,

P represents a data management process, and

Pa represents an application process,

irrespective of the format of the data structure.

In a preferred embodiment the ODPMS may further comprise means for enabling the user to make user defined open associations at run time without the need of a programmer by well defined procedural relations including combinational operators and combination operands defined by the following definitions:

<procedural relation> :: = <combinational operator> <combinational operand>

wherein:

< combinational operator > :: = 'c_1' | 'c_2' | . . . | 'c_n'

< combinational operand > :: = 'd_1' | 'd_2' | . . . | 'd_n' | 'p_1' | 'p_2' | . . . | 'p_n'

wherein $c_1, c_2, \ldots c_n$ represent symbols with preassigned combinational semantics;

and wherein $d_1, d_2, \ldots d_n, p_1, p_2, \ldots p_n$ represent symbols that represent types of data structure variables and types of application program structure

variables,
wherein said user may freely associate any information, text or process at run time via said procedural relations.

According to yet another aspect of the invention there is provided a procedural relation for execution under the control of an open database and preferably process base management system disposed with a computer system comprising a symbolic combinational process required to carry out the procedural relation and a related symbolic structure variable, preferably said combinational process symbol being universal so that the same combinational symbol is used for the same type of structural variable in the system, the combinational process being determined by a combinational operator, and the structure variable being determined by a combinational operand, preferably the process and combinational operators and combinational operands being expressed dynamically at run time as expressed by the following relationships:
<procedural relation> :: = <combinational operator> <combinational operand>
wherein:

< combinational operator > :: = 'c_1' | 'c_2' | . . . | 'c_n'

< combinational operand > :: = 'd_1' | 'd_2' | . . . | 'd_n' | 'p_1' | 'p_2' | . . . | 'p_n'

wherein $c_1, c_2, \ldots c_n$ represent symbols with preassigned combinational semantics;
and wherein $d_1, d_2, \ldots d_n, p_1, p_2, \ldots p_n$ represent symbols that represent types of data structure variables and types of application program structure variables,
wherein said user may freely associate any information, text or process at run time via said procedural relations.

In a preferred embodiment of the invention there is provided a procedural relation processing system for managing a network of distributed heterogeneous databases and distributed heterogeneous process bases, or application systems resident on storage media of at least two distinct computers on the network , at run-time, as they are perceived by the users, directly at the time and place of perception, said procedural relation processing system may further comprise procedural relations which determine, as a unit, a variable combinational process and a

variable structure operand, to which the combinational process is applied, a structure operand being a group of data existing in a database or a group of programming statements existing in a subroutine, or procedure, of an application system, said procedural relation processing system for representing and processing any data or process structure which a user can express as a procedural relation on a terminal of a computer or in an application program, as it is perceived at run-time, directly at the time and place of perception, the procedural relation processing system accepting as input, the procedural relation, wherein said management system after analyzing the semantics of the procedural relation, transfers control to the location on the network where the data exists, identifying the remote application system and what statements to execute, the statements issuing a request to the local Database Management System (DBMS) where the data exists which after receiving the required data values, returns the data values to the user's local area computer and these values are displayed on the terminal by the system.

Preferably said databases and process bases are structured semantically in at least two distinctly different formats (for example hierarchical or relational).

In a preferred embodiment said database is for a single user environment and said data and processes are stored on at least one accessible media available to the single user.

According to yet still another aspect of the invention there is provided a database and preferably a process base management system resident on at least one computer comprising at least a first distributed heterogeneous database and preferably at least a first distributed heterogeneous process base (ODPMS) resident on accessible storage media means, of at least one computer, in a first format, and being accessible to said management system, said management system having user defined schema means including an auxiliary database, process base, text base and screen base pre-defined by a responsible database and process base administrator for said ODPMS and having knowledge of a first format and data model structure and semantic rules of said at least a first distributed heterogeneous database and process base, said process base defining at least one application process which may be performed upon data, text or the like, said application programs being resident with said accessible storage media and being responsive to a call statement, said screen base defining

at least one screen layout for said data and or text to be displayed on display means (such as a CRT) to a user, said text base containing subjective information to be communicated to said user at logical user defined times, said schema defining for a user the ability to pursue procedural relations defined by the following functional relationship:

$$f: (P,D) \longrightarrow \{^{D}_{Pa}$$

where **D** represents a data structure variable,

P represents a data management process, and

Pa represents an application process,

irrespective of format of the data structure.

In a preferred embodiment the ODPMS may further comprise means for enabling the user to make user defined open associations at run time without the need of a programmer by well defined procedural relations including combinational operators and combinational operands defined by the following definitions:

<procedural relation> :: = <combinational operator> <combinational operand>

wherein:

< combinational operator > :: = 'c_1' | 'c_2' | . . . | 'c_n'

< combinational operand > :: = 'd_1' | 'd_2' | . . . | 'd_n' | 'p_1' | 'p_2' | . . . | 'p_n'

wherein $c_1, c_2, \ldots c_n$ represent symbols with preassigned combinational semantics;

and wherein $d_1, d_2, \ldots d_n, p_1, p_2, \ldots p_n$ represent symbols that represent types of data structure variables and types of application program structure variables,

wherein said user may freely associate any information, text or process at run time via said procedural relations.

According to yet still another aspect of the invention there is provided a method for establishing an open database and process base management system for a computer network of distributed heterogeneous databases and process bases for at least one node of said network comprising:

i) preparing a schema, dictionary, data model or the like by a data and process base administrator having full knowledge

of the format, data model, structure and semantic rules of said database and said process base;

ii) defining procedural relations executable by user accessible combinational operators effecting combinational operands; and

iii) reorganizing said information from said database and process base within said schema in a manner to provide clear associations of inter-related information;

wherein preauthorized users are allowed by the use of the user determined procedural relations executed logically at run time to logically and variably create interpretations and associations, query information, and inter-relate information at run time previously unrelatable, unqueriable, and unassociatable because of the structure, data model and semantic rules of the database and/or process base.

According to still yet another aspect of the invention there is provided a method of inter-relating information available in a database contained within at least one computer, and preferably further of inter-relating the processes to which the data may be subjected on said computer comprising:

i) defining the structure or format or relativity of the data available on said database;

ii) defining the schema in a users second computer in an open database and process base management system (ODPMS) after knowing the topography of the accessed database which schema includes defining the processes with which the data on said accessed database will be utilized, the processes being totally user defined and accessible by a call statement;

iii) accessing the information and gating said information into the predefined schema;

iv) manipulating the information based on steps i, ii and iii and developing new relations and inter-relations with the ODPMS of the second computer;

v) executing the application system processes as required based on steps i, ii and iii; and

vi) creating or preparing an association such as a display of the data as seen in steps (iv) and (v) which is accessible to the second computer by the authorized user.

In a preferred embodiment the method may further comprise means for enabling the user to make user defined open associations at run time without the need of a programmer by well defined procedural relations including combinational operators and combinational operands defined by the following definitions:

<procedural relation> :: = <combinational operator> <combinational operand>

wherein:

< combinational operator > :: = 'c_1' | 'c_2' | . . . | 'c_n'

< combinational operand > :: = 'd_1' | 'd_2' | . . . | 'd_n' | 'p_1' | 'p_2' | . . . | 'p_n'

wherein $c_1, c_2, \ldots c_n$ represent symbols with preassigned combinational semantics;

and wherein $d_1, d_2, \ldots d_n, p_1, p_2, \ldots p_n$ represent symbols that represent types of data structure variables and types of application program structure variables.

wherein said user may freely associate any information, text or process at run time via said procedural relations.

In a preferred embodiment the aforementioned method and the ODPMS thereof may further comprise data structures and process structures which logically and variably associate data and processes according to a procedural relation as defined by the following functional relationship:

$$f: (P,D) \longrightarrow \{^{D}_{Pa}$$

where **D** represents a data structure variable,

P represents a data management process, and

Pa represents an application process,

irrespective of the format of the data structure.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention will now be illustrated with reference to the following drawings in which:

Figures 1 and 2 are considered prior art of the invention.

Figure 3 is a schematic view of an open database and process

base model illustrated in a preferred embodiment of the invention.

Figure 4 is a flow diagram and program segment diagram of the relational procedures of the open database and process base model illustrated in a preferred embodiment of the invention.

Figure 5 is a further expansion of Figure 4 illustrating the model record and model segments shown in a preferred embodiment of the invention.

Figures 6 through to 11 refer to an example of the use of the open database and process base model by an unskilled user seeking further information about a Doctor working in a specific hospital and illustrated in a preferred embodiment of the invention. Figure 6 refers to the organization of the database being accessed. Figure 7 refers to a schematic diagram of the network wherein a remote computer is being accessed by a user. Figures 8a and 8b are the screen formats as exhibited on a CRT for the user illustrated in a preferred embodiment of the invention. Figure 9 refers to the data record of the schema of the simple example of Figure 6. Figure 10 refers to another screen layout for a text record shown on a CRT for the user illustrated in a preferred embodiment of the invention. Figure 11 refers to the sets of instructions or logic of the subroutines under which the hospital example operates as shown on a CRT to the user and illustrated in a preferred embodiment of the invention.

Figure 12 provides a process flow diagram of the open database and process base model within the database and process base management system illustrated in a preferred embodiment of the invention.

Figure 13 defines portions S1 and S2 of Figure 12.

Figure 14 describes a general flow chart of the open database and process base model of a database and process base management system illustrated in a preferred embodiment of the invention the flow of the logic thereof.

Figure 15 provides the flow chart of the screen base manager portion of Figure 14 illustrated in a preferred embodiment of the invention.

Figure 16 provides the flow chart of the database manager portion of the general flow chart of Figure 14 illustrated in a preferred embodiment of the invention.

Figure 17 provides the flow chart of the fixed text manager of

the general flow chart of Figure 14 illustrated in a preferred embodiment of the invention.

Figure 18 provides the flow chart of any subroutine of the application system of the general flow chart of Figure 14 illustrated in a preferred embodiment of the invention.

Figures 19 and 20 provide general schematic views of a computer network used in the implementation of a database and process base management system and the data files available respectively as described in the example and illustrated in a preferred embodiment of the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The ODPMS may be implemented by using the language C and the UNIX operating system (please refer to Figure 3).

The Task of the DPBA

The ODPMS manages simultaneously, at run-time:

(a) screen databases,

(b) databases,

(c) text bases, and

(d) process bases,

in any order and in any combination defined at run-time by the users' needs. Thus, the DPBA must design all of these components and their allowable degree of open interaction by the design of:

(a) screen databases

The DPBA designs the format and the text for each screen record. The format and the text for a screen record is open, and determined by the DPBA in Figure 8, which exhibits the screen records necessary for the first embodiment which follows

(b) databases

The DPBA must define for every data file in a database, the names of the data files, their format, and their size. The DPBA also has to define the relational structure(s) for the data file(s) that is (are) necessary, along with the allowable relational (or combinational) interpretations. For example, the data files for the schema records HOSPITAL and DOCTOR in the hierarchical database in City A appears in Figure 9.

(c) text bases

The DPBA determines the contents of the text record in the text base. The number of lines in a text record is variable. A text record

can contain only one line or thousands of lines. In the following embodiment, just two text records whose contents appear in Figure 10 are provided.

(d) process base

The DPBA designs the process base for every site of the network. For the following embodiment, the DPBA has to design two subroutines for site A, which appear in Figure 11.

It appears from the above description, that a DPBA designs and implements the components of the ODPMS in isolation. But this is not the case. All of the structures (screen databases, databases, and text bases), and all of the allowable open interactions among screen databases, databases, text bases, and process bases, along with networking information, is stored symbolically into a file called schema. The Data and Process Definition Language (DPDL) is used by the DPBA to create the schema and all of the components of the ODPMS automatically. A schema is needed for each site if we use the ODPMS.

The function of the DPDL is simply to generate the schema for a particular distributed data and process base application defined by the DPBA.

The DPDL is in reality, an application program for the ODPMS. That is to say, DPDL is a program, (like any other application system), that uses the ODPMS to generate the schema for a particular application as defined above. For this reason, the ODPMS uses a schema (that is, it interprets a schema), which determines how a schema for an application can be generated. Under this scheme, the task of the DPBA is simplified and has a higher degree of correctness. Thus, the DPBA does not specify any instructions for the schema, the DPBA defines data values as required by the DPDL. Thus, the DPDL requires that the DPBA define the following:

i) the names of the end-users for every site of the network (these names must be the login names of the end-users);

ii) the names of the subroutines for every application system and to define the names of the program segments, if any, for every subroutine;

iii) the names of the data files of the database, the size (the number of physical data records) for each file, and the data types for every record type;

iv) the access mechanism for the data files (i.e. direct, random, sequential, etc);

v) whether or not a data file is shareable or non-shareable;

vi the names of the relational files of the database, the size (the number of records) for each file, and the data association interpretation (i.e. hierarchical, relational - which in this case the relational file is empty, or network);

vii) the format and names of every screen record of the screen database for every site;

viii) the screen co-ordinates of the displayed data values;

ix) the names and content for every text record in the text database;

x) the names of the schema records and their associations with subroutines and screen records;

xi) the security and action in case of an error;

xii) the schema segments for every schema record;

xiii) the source that generates the data values for a schema segment;

xiv) the target recipients of these data values (more than one site or more than one component may consume a data segment value;

xv) the allowable procedural relations a user can express;

xvi) the management data structure process and the relational structure file (if any); and

xvii) the operational data structure process and the related data file.

ALL OF THE ABOVE ARE INVISIBLE TO THE USERS

Referring to Figure 4 there is described a process base which is in reality, is an application system. An application system is a set of related subroutines, or procedures. Each subroutine is uniquely identified by a name. This name has been determined by the DPBA. A subroutine is divided into a set of program segments. A program segment is a set of programming statements, identified as a unit, and written in any computer language that has a CALL statement. The semantics of a program segment is open to the DPBA. A subroutine can be shareable among many users, simultaneously, with different combinations of program segments. Please refer to the example program below.

Referring to Figure 5 the ODPM distinctly identifies the

structure variables of the model and the structural interpretation between the structures. The largest structural unit is a model record. A model record is uniquely recognized by a name defined by the DPBA. The relation between two related model records can be fixed or open. When a relation is fixed, we know:

(a) which is the related model record, and

(b) what structural interpretation to assign.

When a relation is open we know what is the domain of the related model records but, we do not know what structural interpretation to assign. This is known after the related model record is identified.

A model record is further divided as a set of disjointed model segments. A model segment is uniquely recognized within a model record by a name defined by the DPBA. A model segment can be related with other model segments that belong in the same model record or with other model records, by fixed or open relations. The same semantics are applied here as before.

A model segment represents the creation and distribution of data values in the entire network and can be defined by the tuple:
<data, creation of data, structural interpretation, distribution of data, procedural relation or relations>
The 'data' determine the data types of the values. For example, integer, real, character, etc. The creation of data determines who generates the data values for this model segment. It can be an end-user, a subroutine or subroutines in the network, or a DBMS in the network. The structural interpretation determines what structural semantics to assign, to identify the data if they are extracted from a database. The distribution of data determines the consumer(s) of the information in the network. The consumers can be end-users, a subroutine, a program segment, or a DBMS, or any combinations of the above.

The procedural relation determines what procedural relations an end-user or a subroutine can express at any place of a model segment. We can explicitly determine the procedural relations assigned to the users. Thus, users with less computer experience may be assigned none or very few procedural relation capabilities until they gain more experience. The DPBA can change this capability at any time. Users with computer knowledge may be assigned the entire procedural relation capability or subsets of it. One must understand that the procedural

relation capability is explicitly determined at the model segment levels. That is, the procedural relation capability for the same user can vary.

How An End-user Extracts Information from the Network

Referring to the Figures 6 through 11, assume that a distributed medical database in the U.S.A exists, where part of it is stored in City A, (or site A), and uses a hierarchical data structure (see Figure 6 and 7). In City B, (or site B), there is an end-user who is connected to the network. This end user, who is sitting in front of a terminal marked with an 'x', in Figure 7 wishes to find out the following information:

> What is the specialty of the doctor who has Doctor Number 1234, and works for the hospital that has Hospital Code H123?

It is extremely difficult, if not impossible, for these questions to be translated into an application program by the end-user, who in this example is a Medical Director. To do so, the Medical Director must know how:

(a) to read schemas,

(b) to convert schema information into application program(s), and

(c) to write the application program(s).

By using the ODPMS, this Medical Director can extract the required information without any help. All they need to know is just one name from the text base.

As discussed, the end-user is a Medical Director who knows nothing about computers, but by using the ODPMS, they can extract any desired information by taking the following steps:

The end-user signs on. The ODPMS starts execution and displays the message,

'PLEASE ENTER THE RECORD'S NAME:' Our end-user knows nothing other than that all of the processing information about the database HOSPITAL is stored in the text record HOSPITAL_INFO. Thus, the end-user types in 'HOSPITAL_INFO'. The ODPMS displays in full screen pages, the text of the file HOSPITAL_INFO. When the end-user is finished with the information, they then type the character 'q', and the process of the ODPMS ends. The end-user puts the ODPMS back on execution. The ODPMS displays the message, 'PLEASE ENTER THE RECORD'S NAME:'.

This time the end-user enters 'HOSPITAL'. The ODPMS puts on execution the screen manager to display the contents of the screen record HOSPITAL, as it appears in Figure 8. Then the ODPMS examines the first schema element of the current schema record which says, transfer control to the screen manager. The screen manager starts execution and examines the next schema element which says, move the cursor at the specified location and wait for input. Thus, the cursor moves to the screen element,

'1) PLEASE ENTER THE HOSPITAL CODE:'

and waits for input. The end-user inserts the name 'H123'. After examining the semantics of the current schema element, the screen manager stores this data value in the data buffer. The screen manager examines the next schema element that says, return control to the ODPMS. The ODPMS examines the next schema elements of the current schema record on execution. The next schema element says, send all the data values of the generated data buffer to site A, and the procedure hospital must be put on execution in site A. The ODPMS sends the information to site A and waits for the return from site A.

In site A, the procedure hospital starts execution. The procedure hospital calls the local the ODPMS. Although in the argument list of the call statement many variables exist, the ODPMS does not process the entire argument list at once, but it processes one schema segment at a time. Thus, if the end-user calls the same procedure immediately, or at a later time, the ODPMS will execute the next schema segment, which may not be in sequence with the previous. The ODPMS in turn, puts the DBMS of site A on execution. The DBMS examines the next schema element of the schema record which has the semantics, this data value is a key value for the data file HOSPITAL; Search the data file HOSPITAL sequentially, one does not care about the access mechanism, instead the ODPMS emphasizes the interpretation of the access mechanism. The DBMS finds the data record that has this key. In turn, the DBMS examines the next schema element that says, get the name of the hospital and put it in the data buffer of the procedure hospital. The DBMS examines the next schema element that says, return to the application program. In turn, the application program transfers control to site B, with the name of the hospital.

In site B, the ODPMS examines the next schema element that

says, transfer control to the screen manager. The screen manager examines the next schema element that says, get the value from the data buffer and display it on the screen at the location

'2) HOSPITAL NAME:'

After examining the next schema element, the screen manager, waits for input from the end-user. The end-user does not know what to do, so they enter 'HOSPITAL_INFO'. The screen manager stacks the state of the ODPMS, clears the physical screen, and starts to display the contents of the text record HOSPITAL_INFO of the text base as it appears in Figure 10. The screen manager, as before, displays in full screen pages, the text of the file HOSPITAL_INFO. When the end-user is finished with the information, they then type the character 'q'. The screen manager restores the state of the ODPMS. Thus, the screen manager waits for input from the end-user. This time they enter:

'<DOCTOR'

This symbol means, put the screen record DOCTOR on the physical screen, as it appears in Figure 8. The screen manager, again, stacks the state of the ODPMS, clears the screen, and then displays the screen record DOCTOR, and returns control to the ODPMS. The ODPMS examines the first schema element of the new schema record which says, transfer execution control to the screen manager. The screen manager examines the next schema element that says, move the cursor to the screen location

'1) DOCTOR NUMBER:'

and waits for input and after the correct input is inserted, stores it in the data buffer. The end-user inserts the value '1234'. After examining the next schema element, the screen manager returns control to the ODPMS. The ODPMS examines the next schema element that says, send all the data values of the generated data buffer to site A and the procedure doctor must be put on execution. The ODPMS sends the information to site A and waits for the return from site A.

In site A, the procedure doctor starts execution. The procedure doctor calls the local ODPMS. The ODPMS of site A puts the DBMS of site A on execution. The DBMS examines the next schema element that says, the current hospital data record will be used as the head of the list to find the required doctor. Thus, the DBMS finds the corresponding data record in the relational structure A. The contents of this data record determines the list of the doctors that work for the

hospital H123. The DBMS processes every doctor's number in the data file DOCTOR, in the order specified by the relational structure B. When the DBMS finds the desired record, it stores the doctor's name and specialty in the data buffer, as before. Then the DBMS examines the next schema element that says, return control to the application program. The application program transfers control to site B with the doctor's name and specialty.

In site B, the ODPMS examines the next schema element and as before, transfers control to the screen manager, which in turn, displays these two data values on the screen. After that, the screen manager moves the cursor back to the screen element,

'1) DOCTOR NUMBER:'

and waits for the user to enter a new DOCTOR NUMBER value.

Let's assume that the end-user does not know what to do next. They can enter the character ';' , instead of a DOCTOR NUMBER. Now the screen manager saves the state of the ODPMS and after clearing the physical screen, it displays the run-time symbols of the ODPMS (see examples below). The end-user finds out that the symbol '~' terminates the process of any screen record, at any place. Thus, the end-user enters the symbol '~', and the screen record DOCTOR is deleted from execution. Thus, the screen is cleared and the screen record HOSPITAL appears on the screen exactly at the stage that it was previously. That is, the screen manager expects input from the end-user, who inserts again, the symbol '~', and the screen record HOSPITAL is deleted from execution, and the DBMS of site A deletes the data record HOSPITAL from the database.

A concise listing of examples of procedural relations and operators follows. This list is by no means exhaustive, and is defined by the ODPMS.

RUN-TIME SYMBOLS	SEMANTICS OR MEANING
#	Display input data type. If it is numeric display the range; if it is a string, display the number of characters.
"	For the current screen element accept the previous values that exist in the data buffer.
@	Display all the associated record types.
$	Display all the associated application

	processes.
( )	Clear the entire screen display.
:	Move to an empty screen row and accept long input.
;	Display all the run-time symbols used by the ODPMS and their semantics.
?	Ignore the entire input in the current screen record and repeat the process from the beginning.
.	Accept missing values for the current screen element.
^	Decrease the screen record counter by 1 and repeat the process.
!	Move to the beginning of the current screen segment.
&n (n is an integer number)	Move the cursor to the n^{th} screen element of the current screen record.
~	Terminate the entire process of the current screen record and transfer control to the previous one (if any).
< string (string is a set of letters, numbers, underscores)	Transfer control to the screen record identified by the string.
<< string	Transfer control to the screen record identified by the string and use the key value(s) that exist in the data buffers.
> string	Transfer control to the subroutine (of the application system) identified by the string.

ANOTHER EXAMPLE

Let's assume that a user of a network system is using terminal X, as it appears in Figure 19 to access data at location L2 as seen in Figure 20.

The user signs on. A logical screen record is displayed on the CRT. The ODPMS prompts the user to enter the value for a screen element a. The user enters the value for a screen element a. The ODPMS

then transfers control to location L2, where the data file A exists. The proper record is accessed and the value b is extracted and displayed on the CRT of location L1. Continuing the process of the screen record, the ODPMS finds the values for the screen element c, d, and e which also exist in location L2 in the data file B. Moreover, these values are associated with the previous values (a and b) and the association is presented in a relational data structure. When the ODPMS gets the values for c, d and e, they are displayed on the CRT. Although the user may not have completed the whole screen record values at this point, (or any other point), they may dictate how the logical process can continue. For example:

i) the user may repeat the same cycle a, b, c, d, and e for different values of a, or

ii) the user may stop the process of this screen record, or

iii) the user may extract descriptive information, or fixed text about the semantics of a data value, or

iv) the user may wish to proceed with the remaining process of this screen record.

If this is the case, the ODPMS may prompt the user to assign values for the data value f. The proper physical record of the data file C at location L3 is found and the value for g and h are extracted and displayed by the ODPMS.

After this, the ODPMS continues the process of the screen elements i and j whose values exist at location L3 in the data file D and are associated with the previous values f, g and h in a hierarchical data structure. If more than one set of i and j exists, the ODPMS will transfer control to the next screen element which in reality is a control element. If the user replies 'yes' and more data values exist for i and j, the process is repeated. If there are no more values for i and j or the user replies 'no', the screen record process is repeated (i.e. the ODPMS prompts the user for a value of the screen element a). The user stops the process by inserting the symbolic string '~' for the value of the screen element a.

In other DBMSs, a logical data view must be known, as a whole, by the DBMS at compilation time. Under the ODPMS, the logical data view is unbounded. That is, it can be increased or decreased at run time by ways that are not foreseen or obvious to the user.

The details of this following flow chart in Figures 12-18 are

self evident. Figure 12 provides a process flow diagram of the open database and process base model within the database and process base management system. Figure 13 defines portions S1 and S2 of Figure 12. Figure 14 describes a general flow chart of the open database and process base model of a database and process base management system and the flow of the logic thereof. Figure 15 provides the flow chart of the screen base manager portion of Figure 14. Figure 16 provides the flow chart of the database manager portion of the general flow chart of Figure 14. Figure 17 provides the flow chart of the fixed text manager of the general flow chart of Figure 14. Figure 18 provides the flow chart of any subroutine of the application system of the general flow chart of Figure 14.

The following example of an application program to implement the Open Data and Process Base Model (ODPMS) is provided. This program is written in the language C and is provided only as an example of the manner in which the Open Data and Process Base Management System may be implemented in one embodiment. This example describes how the Screen Database, Text Base and Process Base are implemented but does not provide details on a DBMS or a client server. Those skilled in the art will appreciate that the following program is therefore only an example, after providing all of the necessary definitions of the routines which are used to effect the procedural relations. Other applications may be equally acceptable providing they follow the logic described in the summary of the invention and throughout the examples of the detailed description of the preferred embodiment.

The following represents a Real Estate Application of the ODPMS for various properties within a community and the recording of changes in ownerships including pricing information:

```
#include <stdio.h>
/****** THIS IS THE APPLICATION PROGRAM TO ODPMS ******,
/********** THE NAME OF THE FILE IS: "application.c" *********/

struct record1
{
        int prop....id;
        char address1 [41];
        char address2 [41];
        char city [26];
        char postal....code [7];
        char prop....name [41];
        char prop....type [5];
        int prop....sqft;
        int prop....flrs;
        int year....built;
        int 1stsale....id;
        int owner....id;
};

struct record2
{
        int sale....id;
        int prop....id;
        int buyer....id;
        int sellor....id;
        int rep....id;
        float asked....amt;
        float sold....amt;
        char date [9];
        char sale....dt [9];
        char close....dt [9];
        char comment [35];
};
```

```
struct record3
{
        int person....id;
        char per....fg [2];
        char owner....fg[2];
        char buyer....fg[2];
        char sellor....fg[2];
        char first....nm[17];
        char last....nm[17];
        char honourific[6];
        char position[41];
        char company....nm[41];
        char phone....bus[11];
        char phone....car[11];
        char phone....fax[11];
        char phone....res[11];
};

struct record4
{
        int sale....id;
        char address1[41];
        char address2[41];
        char city[26];
        char postal....code[7];
        char first....nm[17];
        char last....nm[17];
        char phone....bus[11];
        char phone....car[11];
        char phone....fax[11];
        char phone....res[11];
        char prop....type[5];
        int prop....sqft;
        int year....built;
        float asked....amt;
        float sold....amt;
        char place....dt[9];
```

```
        char sale....dt[9];
        char close....dt[9];
        char comment [35];
};

        struct record5
{
        int link....text;
        char comment[35];
};

        struct record6
{
        int last....sale....id;
        int last....prop....id;
        int last....person....id;
        int last....notes....id;
};

int     add....property(), dis....property(), mod....property(), del....property(),
        add....propsale(), dis....propsale(), mod....propsale(), del....propsale(),
        add....person(), dis....person(), mod....person(), del....person(),
        dis....control(), dis....notes(), mod....notes(),
        add....retsales(), dis....retsales(), mod....retsales(), del....retsales(),
        directory....1(), directory....2();
int     (*function[]) () =
        {add....property, dis....property, mod....property, del....property,
        add....propsale, dis....propsale, mod....propsale, del....propsale,
        add....person, dis....person, mod....person, del....person,
        dis....control, dis....notes, mod....notes,
        add....retsales, dis....retsales, mod....retsales, del....retsales,
        directory....1, directory....2};

main()
{
        void ODPMS();
        char record....name[16];
```

```
        int psv[10];

        printf("\n\n\n PLEASE ENTER THE RECORD'S NAME:");
        scanf("%s", record....name);

        /********** CALL ODPMS **********/

        ODPMS (psv,"DUMMY", record....name);
        if (psv[4] < 0) exit (0);
        (*function[psv[2]]) (); /* CALL THE PROPER PROCEDURE */
}       /**** ENF OF main () ****/

directory....1()
{
        int psv[10];
        while(1)
        {
                ODPMS(psv,"DUMMY", "DUMMY");
                if(psv[4] < 0) return;
                (*function[psv[2]]) (); /**** CALL THE PROPER PROCEDURE
        ****/
        }
}

directory....2()
{
        int psv[10];
        while(1)
        {
                ODPMS (psv, "DUMMY", "DUMMY");
                if(psv[4] < 0) return;
                (*function[psv[2]]) (); /**** CALL THE PROPER PROCEDURE
        ****/
        }
}
```

```
add....property()
{
        struct record1 property;
        char record....name[16];
        int psv[10], return....value;
        /********* CALL ODPMS **********/
        while(1)
        {
                ODPMS(psv, & property,record....name);
                if(psv[4] < 0) return;
                if(psv[5] = = 8) /** Call function **/
                {
                        (*function[psv[2]]) ();
                        continue;
                }
                if (psv[6] = = 0)
                {
                        return....value=disproperty(property.prop....id,
                        &property);
                        if (return....value = = 0)
                        {
                                psv[0] = 101;
                                psv[1] = 0;
                                continue;
                        }
                        continue;
                }
        if (psv[8] = = 12)
        {
                return....value=addproperty(property.prop....id,property);
                return....value=addcontrol(2);
        }
    }
}
```

```
dis....property()
{
    struct record1 property;
    char record....name[16];
    int psv[10],return....value;
    /********* CALL ODPMS **********/
    while(1)
    {
        ODPMS(psv, &property,record....name);
        if(psv[4] < 0) return;
        if(psv[5] = = 8) /** call function **/
        {
                (*function[psv[2]]) ();
                continue;
        }
        if(!psv[6])
        {
                return value = disproperty(property.prop....id,&property);
                if (return....value)
                {
                        psv[0] =101;
                        psv[1] = 1;
                }
            }
        }
    }

mod...property()
{
        struct record1 property;
        char record....name[16];
        int psv[10],return....value;
    /********* CALL ODPMS **********/
    while(1)
    {
        ODPMS(psv, &property,record....name);
        if(psv[4] < 0) return;
```

```
            if(psv[5] = = 8) /** call function **/
            {
                    (*function[psv[2]]) ();
                    continue;
            }
            if(psv[8] <12)
            {
                    return....value = disproperty(property.prop....id,&property);
                    if(return....value)
                    {
                            psv[0] = 101;
                            psv[1] = 1;
                            continue;
                    }
                    continue;
    }

    if(psv[8] = = 12)
        return....value = modproperty(property.prop....id, property);
        }
    }

    del....property()
    {
        struct record1 property;
        char record....name[16];
        int psv[10],return value;
        /********* CALL ODPMS **********/
        while(1)
        {
            ODPMS(psv, &property,record....name);
            if(psv[4] < 0) return;
            if(psv[5] = = 8) /** call function **/
            {
                    (*function[psv[2]]) ();
                    continue;
            }
```

```
        if(psv[8] <12)
        {
                return....value = disproperty(property.prop....id,&property);
                if(return....value)
                {
                        psv[0] = 101;
                        psv[1] = 1;
                        continue;
                }
                continue;
    }

        if(psv[8] = = 12)
        {
                return....value = delproperty(property.prop....id);
                return....value = delcontrol(2);
        }
    }
}

add....propsale()
{
    struct record2 propsale;
    char record....name[16];
    int psv[10],return....value;
    /********* CALL ODPMS **********/
    while(1)
    {
        ODPMS(psv, &propsale,record....name);
        if(psv[4] < 0) return;
        if(psv[5] = = 8) /** call function **/
        {
            (*function[psv[2]]) ();
            continue;
        }
        if(psv[6] = = 0)
        {
```

```
                return....value = dispropsale(propsale.sale....id,&propsale);
                if(return....value = = 0)
                {
                        psv[0] = 101;
                        psv[1] = 0;
                        continue;
                }
                continue;
        }
        if(psv[8] = = 11)
        {
                return....value = addpropsale(propsale.sale....id, propsale);
                return....value = addcontrol(1);
        }
    }
}

dis....propsale()
{
    struct record2 propsale;
    char record....name[16];
    int psv[10],return....value;
    /********* CALL ODPMS **********/
    while(1)
    {
        ODPMS(psv, &propsale,record....name);
        if(psv[4] < 0) return;
        if(psv[5] = = 8) /** call function **/
        {
                (*function[psv[2]]) ();
                continue;
        }
        if(!psv[6])
        {
                return....value = dispropsale(propsale.sale....id,&propsale);
                if(return....value)
                {
```

```
                    psv[0] = 101;
                    psv[1] = 1;
                }
            }
        }
    }

    mod....propsale()
    {
        struct record2 propsale;
        char record....name[16];
        int psv[10],return....value;
        /********* CALL ODPMS **********/
        while(1)
        {
            ODPMS(psv, &propsale,record....name);
            if(psv[4] < 0) return;
            if(psv[5] = = 8) /** call function **/
            {
                (*function[psv[2]]) ();
                continue;
            }
            if(psv[8] < 11)
            {
                return....value = dispropsale(propsale.sale....id,&propsale);
                if(return....value)
                {
                    psv[0] = 101;
                    psv[1] = 1;
                    continue;
                }
                continue;
            }
            if(psv[8] = = 11)
                return....value = modpropsale(propsale.sale....id, propsale);
            }
        }
```

```
del....propsale()
{
    struct record2 propsale;
    char record....name[16];
    int psv[10],return....value;
    /********* CALL ODPMS **********/
    while(1)
    {
        ODPMS(psv, &propsale,record....name);
        if(psv[4] < 0) return;
        if(psv[5] = = 8) /** call function **/
        {
                (*function[psv[2]]) ();
                continue;
        }
        if(psv[8] < 11)
        {
                return....value = dispropsale(propsale.sale....id,&propsale);
                if(return....value)
                {
                        psv[0] = 101;
                        psv[1] = 1;
                        continue;
                }
                continue;
        }
        if(psv[8] = = 11)
        {
                return....value = delpropsale(propsale.sale....id);
                return....value = delcontrol(1);
        }
    }
}
```

```
add....person()
{
    struct record3 person;
    char record....name[16];
    int psv[10],return....value;
    /********* CALL ODPMS **********/
    while(1)
    {
        ODPMS(psv, &person,record....name);
        if(psv[4] < 0) return;
        if(psv[5] = = 8) /** call function **/
        {
                (*function[psv[2]]) ();
                continue;
        }
        if(psv[6] = = 0)
        {
                return....value = disperson(person.person....id,&person);
                if(return....value = = 0)
                {
                        psv[0] = 101;
                        psv[1] = 0;
                        continue;
                }
                continue;
        }
        if(psv[8] = = 14)
        {
                return....value = addperson(person.person....id,person);
                return....value = addcontrol(3);
        }
    }
}
```

```
dis....person()
{
    struct record3 person;
    char record....name[16];
    int psv[10],return....value;
    /********* CALL ODPMS **********/
    while(1)
    {
        ODPMS(psv, &person,record....name);
        if(psv[4] < 0) return;
        if(psv[5] = = 8) /** call function **/
        {
                (*function[psv[2]]) ();
                continue;
        }
        if(!psv[6])
        {
                return....value = disperson(person.person....id,&person);
                if(return....value)
                {
                        psv[0] = 101;
                        psv[1] = 1;
                }
        }
    }
}

mod....person()
{
    struct record3 person;
    char record....name[16];
    int psv[10],return....value;
    /********* CALL ODPMS **********/
    while(1)
    {
        ODPMS(psv, &person,record....name);
        if(psv[4] < 0) return;
```

```
        if(psv[5] = = 8) /** call function **/
        {
                (*function[psv[2]]) ();
                continue;
        }
        if(psv[8] < 12)
        {
                return....value = disperson(person.person....id, &person);
                if(return....value)
                {
                        psv[0] = 101;
                        psv[1] = 1;
                        continue;
                }
                continue;
        }
        if(psv[8] = = 14)
                return....value = modperson(person.person....id, person);
        }
    }

    del....person()
    {
        struct record3 person;
        char record....name[16];
        int psv[10],return....value;
        /********* CALL ODPMS **********/
        while(1)
        {
            ODPMS(psv, &person,record....name);
            if(psv[4] < 0) return;
            if(psv[5] = = 8) /** call function **/
            {
                    (*function[psv[2]]) ();
                    continue;
            }
            if(psv[8] < 12)
```

```
        {
                return....value = disperson(person.person....id, &person);
                if(return....value)
                {
                        psv[0] = 101;
                        psv[1] = 1;
                        continue;
                }
                continue;
        }
        if(psv[8] = = 14)
        {
                return....value = delperson(person.person....id);
                return....value = delcontrol(3);
        }
    }
}

dis....control()
{
    struct record6 control;
    char record....name[16];
    int psv[10],return....value;
    /********* CALL ODPMS **********/
        return....value = discontrol( &control);
        while(1)
    {
        ODPMS(psv, &control,record....name);
        if(psv[4] = =-1) return;
    }
}
```

```
dis....notes()
{
    struct record5 notes;
    char record....name[16];
    int psv[10],return....value;
    /********* CALL ODPMS **********/
    while(1)
    {
        ODPMS(psv, &notes,record....name);
        if(psv[8] = = 1) return....value = disnotes( & notes);
        if(psv[5] = = 8) /** call function **/
        {
                (*function[psv[2]]) ();
                continue;
        }
        if(psv[4]  = = -1) return;
    }
}

mod....notes()
{
}

add....retsales()
{
    struct record4 retsale;
    char record....name[16];
    int psv[10],return....value;
    /********* CALL ODPMS **********/
    while(1)
    {
        ODPMS(psv, &retsale,record....name);
        if(psv[4] < 0) return;
        if(psv[5] = = 8) /** call function **/
        {
                (*function[psv[2]]) ();
                continue;
```

```
            }
            if(psv[6] = = 0)
            {
                    return....value = disretsale(retsale.sale....id,&retsale);
                    if(return....value = = 0)
                    {
                            psv[0] = 101;
                            psv[1] = 0;
                            continue;
                    }
                    continue;
            }
            if(psv[8] = = 20)
            {
                    return....value = addretsale(retsale.sale....id, retsale);
                    return....value = addcontrol(1);
            }
        }
    }

    dis....retsales()
    {
        struct record4 retsale;
        char record....name[16];
        int psv[10],return....value;
        /********* CALL ODPMS **********/
        while(1)
        {
            ODPMS(psv, &retsale,record....name);
            if(psv[4] < 0) return;
            if(psv[5] = = 8) /** call function **/
            {
                    (*function[psv[2]]) ();
                    continue;
            }
            if(!psv[6])
            {
```

```
                return....value = disretsale(retsale.sale....id,&retsale);
                if(return....value)
                {
                        psv[0] = 101;
                        psv[1] = 1;
                }
        }
    }
}

mod....retsales()
{
    struct record4 retsale;
    char record....name[16];
    int psv[10],return....value;
    /********* CALL ODPMS **********/
    while(1)
    {
        ODPMS(psv, &retsale,record....name);
        if(psv[4] < 0) return;
        if(psv[5] = = 8) /** call function **/
        {
                (*function[psv[2]]) ();
                continue;
        }
        if(psv[8] < 12)
        {
                return....value = disretsale(retsale.sale....id, &retsale);
                if(return....value)
                {
                        psv[0] = 101;
                        psv[1] = 1;
                        continue;
                }
                continue;
        }
        if(psv[8] = = 20)
```

```
                return....value = modretsale(retsale.sale....id, retsale);
            }
        }

5   del....retsales()
    {
        struct record4 retsale;
        char record....name[16];
        int psv[10],return....value;
10      /********* CALL ODPMS **********/
        while(1)
        {
            ODPMS(psv, &retsale,record....name);
            if(psv[4] < 0) return;
15          if(psv[5] = = 8) /** call function **/
            {
                    (*function[psv[2]]) ();
                    continue;
            }
20          if(psv[8] < 12)
            {
                    return....value = disretsale(retsale.sale....id, &retsale);
                    if(return....value)
                    {
25                          psv[0] = 101;
                            psv[1] = 1;
                            continue;
                    }
                    continue;
30          }
            if(psv[8] = = 20)
            {
                    return....value = delretsale(retsale.sale....id);
                    return....value = delcontrol(1);
35          }
        }
    }
```

As many changes can be made to the preferred embodiments of the invention without departing from the scope of the invention; it is intended that all material contained herein be interpreted as illustrative of the invention and not in a limiting sense.

THE EMBODIMENTS OF THE INVENTION IN WHICH AN EXCLUSIVE PROPERTY OR PRIVILEGE IS CLAIMED ARE AS FOLLOWS:

1. An Open Data and Process Base Management System (ODPMS) for a computer installed network of heterogeneous distributed databases and process bases resident on computing means comprising

- means for preparing a schema by the Database Process Base Administrator (DBPA) for preferably each local ODPMS so as to define a screen base manager, a database manager, a text base manager, and a process base manager, and the manner in which control may be moved to and from subschema elements by the ODPMS
- means for providing procedural relations allowing a user to access and associate data or processes or text as desired from one or more databases or process bases at run time via combinational operators and operands, preferably as per the following relationship:

<procedural relation> :: = <combinational operator> <combinational operand>
wherein:

< combinational operator > :: = 'c_1 ' | 'c_2' | . . . | 'c_n'

< combinational operand > :: = 'd_1 | 'd_2' | . . . | 'd_n' | 'p_1' | 'p_2' | . . . | 'p_n'

wherein $c_1, c_2 \ldots c_n$ represent symbols with preassigned combinational semantics;

and wherein $d_1, d_2 \ldots d_n, p_1, p_2, \ldots p_n$ represent symbols that represent types of data structure variables and types of application program structure variables,
wherein said user may freely associate any information, text or process at run time via said procedural relations;

- means for circumventing the data model or schema of the heterogeneous distributed database or process base to therefore provide data, or processes responsive to a call statement, as desired by the user under control of the ODPMS
- whereby the user may, if authorized to do so in the schema, logically

variably associate data and or text and execute processes on data, from databases and process bases with dissimilar, data models, semantics, and protocol on an open variable user defined query basis.

2. Means for accessing data resident on at least one computer in at least one database and preferably processes in at least one process base, of a known structure (for example hierarchical, network, or relational) comprising:

- schema means defined by a Data and Process Base Administrator (DBPA) for providing a database, process base, text base and screen base,
- means for associating information in a manner outside of the structure and protocol or rules of said database on a logical variable open basis without the need for a specific Application Program Interface,
- means for enabling the user to make user defined open associations at run time without the need of a programmer by well defined procedural relations including combinational operators and combinational operands defined by the following definitions:

<procedural relation> :: = <combinational operator> <combinational operand>
wherein:

< combinational operator > :: = 'c_1 ' | 'c_2' | . . . | 'c_n'

< combinational operand > :: = 'd_1 | 'd_2' | . . . | 'd_n' | 'p_1' | 'p_2' | . . . | 'p_n'

wherein $c_1, c_2, \ldots c_n$ represent symbols with preassigned combinational semantics;
and wherein $d_1, d_2, \ldots d_n, p_1, p_2, \ldots p_n$ represent symbols that represent types of data structure variables and types of application program structure variables,
wherein said user may freely associate any information, text or process at run time via said procedural relations.

3. An Open Data and Process Base Management System (ODPMS) preferably for use with a network of heterogeneous distributed databases and process bases, said ODPMS comprising data structures and

process structures which logically and variably associate data and processes according to a procedural relation as defined by the following functional relationship:

$$f: (P,D) \longrightarrow \{^{D}_{Pa}$$

where **D** represents a data structure variable,
P represents a data management process, and
Pa represents an application process,
irrespective of the format of the data structure.

4. A procedural relation for execution under the control of an open database and preferably process base management system disposed with a computer system comprising a symbolic combinational process required to carry out the procedural relation and a related symbolic structure variables, preferably said combinational process symbol being universal so that the same combinational symbol is used for the same type of structural variable in the system, the combinational process being determined by a combinational operator, and the structure variable being determined by a combinational operand, preferably the process and combinational operators and combinational operands being expressed dynamically at run time as expressed by the following relationships:

<procedural relation> :: = <combinational operator> <combinational operand>
wherein:

< combinational operator > :: = 'c_1 ' | 'c_2' | . . . | 'c_n'

< combinational operand > :: = 'd_1 | 'd_2' | . . . | 'd_n' | 'p_1' | 'p_2' | . . . | 'p_n'

wherein $c_1, c_2, \ldots c_n$ represent symbols with preassigned combinational semantics;
and wherein $d_1, d_2, \ldots d_n, p_1, p_2, \ldots p_n$ represent symbols that represent types of data structure variables and types of application program structure variables,
wherein said user may freely associate any information, text or process at

run time via said procedural relations.

5. The system of claim 1-4 wherein said databases and process bases are structured semantically in at least two distinctly different formats (for example hierarchical or relational).

6. The system of claim 1-4 wherein said database is for a single user environment and said data and processes are stored on at least one accessible media available to the single user.

7. An open database and preferably a process base management system resident on at least one computer comprising at least a first distributed heterogeneous database and preferably at least a first distributed heterogeneous process base, (ODPMS) resident on accessible storage media means, of at least one computer, in a first format, and being accessible to said management system, said management system having user defined schema means including an auxiliary database manager, process base manager, text base manager, screen base manager pre-defined by a responsible database and process base administrator for said ODPMS and having knowledge of a first format and data model structure and semantic rules of the at least a first distributed heterogeneous database and process base, said process base defining at least one application process which may be performed upon data, text or the like, said application process being resident with said accessible storage media and being responsive to a call statement, said screen base defining at least one screen layout for said data and or text to be displayed on display means (such as a CRT) to a user, said text base containing subjective information to be communicated to said user at logical user defined times, said schema defining for a user the ability to pursue procedural relations defined by the following functional relationship:

$$f: (P,D) \longrightarrow \{^{D}_{Pa}$$

where **D** represents a data structure variable,
P represents a data management process, and
Pa represents an application process,
irrespective of format of the data structure.

8. A method for establishing an Open Data and Process Base Management System for a computer network of distributed heterogeneous databases and process bases for at least one node of said network comprising:

i) preparing a schema, dictionary, data model or the like by a data and process base administrator having full knowledge of the format, data model, structure and semantic rules of said database and said process base being accessed;

ii) defining procedural relations executable by user accessible combinational operators effecting combinational operands; and

iii) reorganizing said information from said database and process base within said schema in a manner to provide clear associations of inter-related information,

wherein preauthorized users are allowed by the use of the user determined procedural relations executed logically at run time to logically and variably create interpretations and associations, query information, and inter-relate information at run time previously unrelatable, unqueriable, and unassociatable because of the structure, data model and semantic rules of the database and/or process bases.

9. A method of inter-relating information available in an accessible database contained within at least one computer, and preferably further of inter-relating the processes to which the data may be subjected on said computer comprising:

i) defining the structure or format or relativity of the data available on said database;

ii) defining a schema in a users second computer in an Open Data and Process Base Management System (ODPMS) after knowing the topography of the accessed database, which schema includes defining the processes with which the data on said accessed database will be utilized, the processes being totally user defined and accessible by a call statement;

iii) accessing the information and gating said information into

the predefined schema;

iv) manipulating the information based on steps i, ii and iii and developing new relations and inter-relations with the ODPMS of the second computer;

v) executing the application system processes as required based on steps i, ii and iii; and

vi) creating or preparing an association such as a display of the data based on steps (iv) and (v), which is accessible to the second computer by the authorized user.

10. The ODPMS of claim 3 or 7 further comprising means for enabling the user to make user defined open associations at run time without the need of a programmer by well defined procedural relations including combinational operators and combinational operands defined by the following definitions:

<procedural relation> :: = <combinational operator> <combinational operand>
wherein:

< combinational operator > :: = 'c_1 ' | 'c_2' | . . . | 'c_n'

< combinational operand > :: = 'd_1 | 'd_2' | . . . | 'd_n' | 'p_1' | 'p_2' | . . . | 'p_n'

wherein $c_1, c_2, \ldots c_n$ represent symbols with preassigned combinational semantics;
and wherein $d_1, d_2, \ldots d_n, p_1, p_2, \ldots p_n$ represent symbols that represent types of data structure variables and types of application program structure variables,
wherein said user may freely associate any information, text or process at run time via said procedural relations.

11. The method of claim 8 or 9 further comprising means for enabling the user to make user defined open associations at run time without the need of a programmer by well defined procedural relations including combinational operators and combinational operands defined by the following definitions:

<procedural relation> :: = <combinational operator> <combinational operand>
wherein:

< combinational operator > :: = 'c_1 ' | 'c_2' | . . . | 'c_n'

< combinational operand > :: = 'd_1 | 'd_2' | . . . | 'd_n' | 'p_1' | 'p_2' | . . . | 'p_n'

wherein $c_1, c_2 \ldots c_n$ represent symbols with preassigned combinational semantics;

and wherein $d_1, d_2 \ldots d_n, p_1, p_2, \ldots p_n$ represent symbols that represent types of data structure variables and types of application program structure variables,
wherein said user may freely associate any information, text or process at run time via said procedural relations.

12. The invention of claim 1, 2, 4 or 10 further comprising a procedural relation processing system for managing a network of distributed heterogeneous databases and distributed heterogeneous process bases, or application systems resident on storage media of at least two distinct computers on the network, at run-time, as they are perceived by the users, directly at the time and place of perception, said procedural relation processing system comprising procedural relations which determine, as a unit, a variable combinational process and a variable structure operand, to which the combinational process is applied, a structure operand being a group of data existing in a database or a group of programming statements existing in a subroutine, or procedure, of an application system, said procedural relation processing system for representing and processing any data or process structure which a user can express as a procedural relation on a terminal of a computer or in an application program, as it is perceived at run-time, directly at the time and place of perception, the procedural relation processing system accepting as input, the procedural relation, wherein said management system after analyzing the semantics of the procedural relation, transfers control to the location on the network where the data exists, identifying the remote

application system and what statements to execute, the statements issuing a request to the local Database Management System (DBMS) where the data exists which after receiving the required data values, returns the data values to the user's local area computer and these values are displayed on the terminal by the system.

13. The method of claim 8, 9 or 11 wherein said ODPMS further comprises data structures and process structures which logically and variably associate data and processes according to a procedural relation as defined by the following functional relationship:

$$f: (P,D) \longrightarrow \{^{D}_{Pa}$$

where **D** represents a data structure variable,
P represents a data management process, and
Pa represents an application process,
irrespective of the format of the data structure.

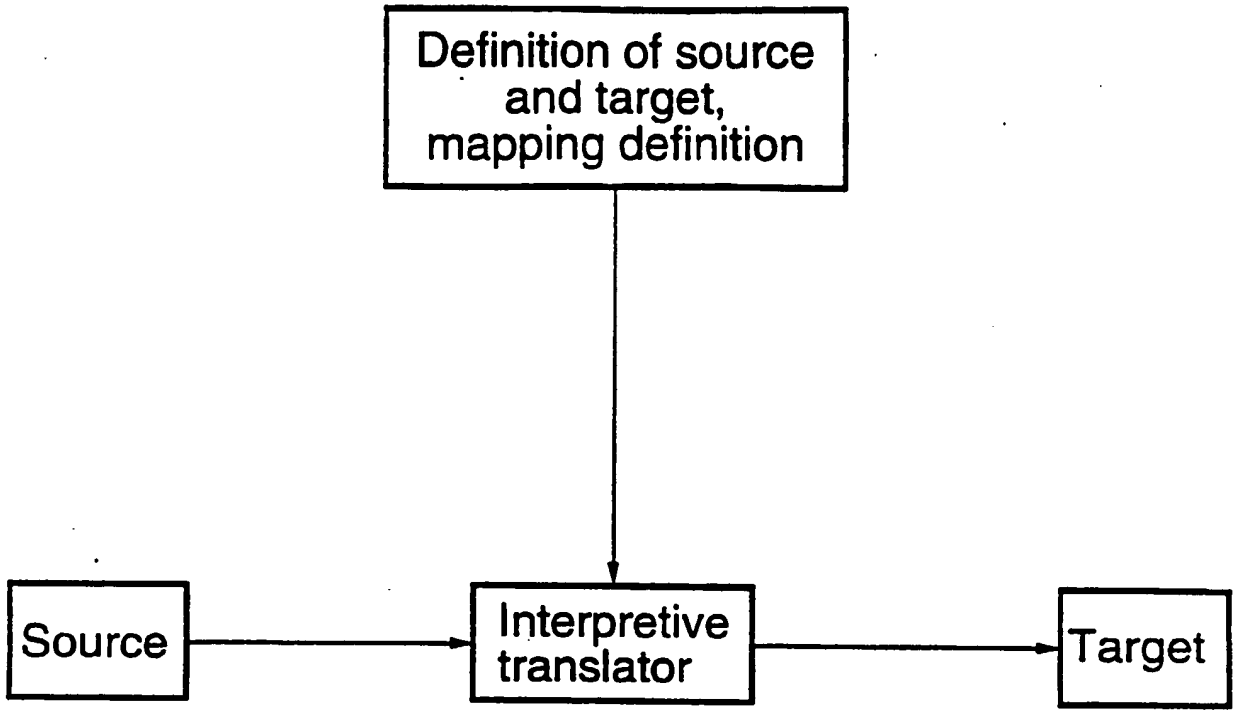

Definition of source and target, mapping definition
Source
Interpretive translator
Target

FIG.1.

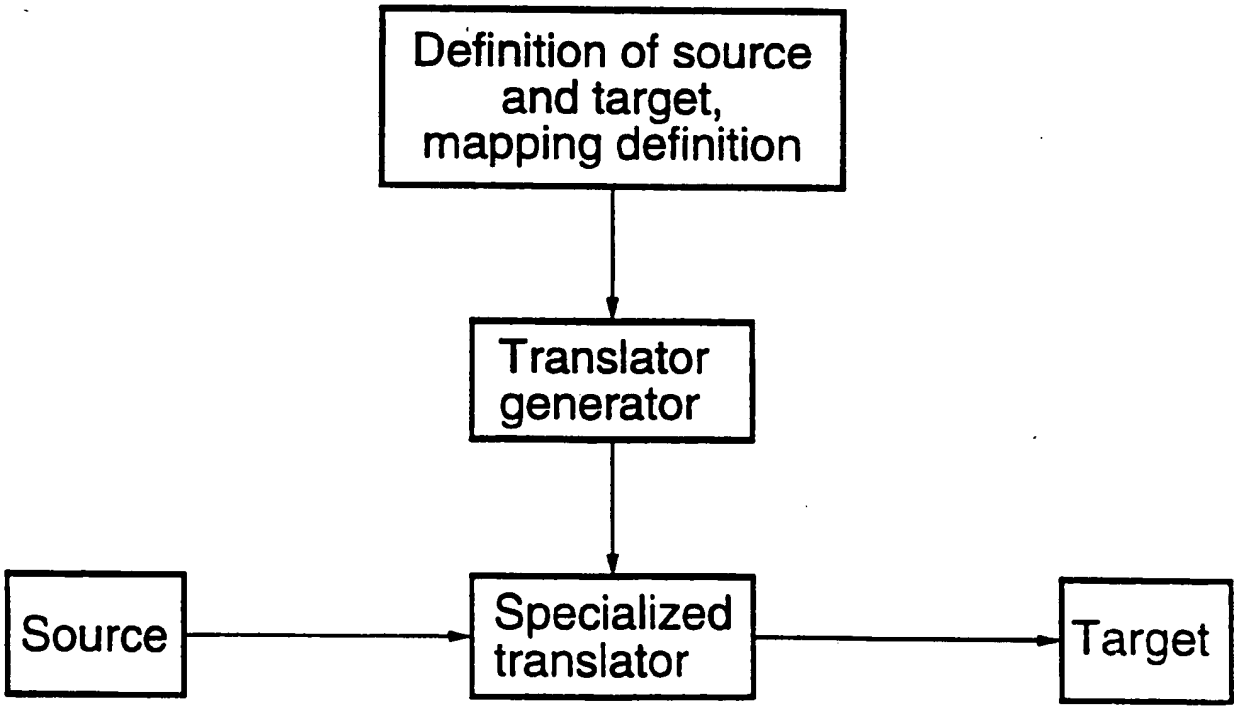

Definition of source and target, mapping definition
Translator generator
Source
Specialized translator
Target

FIG.2.

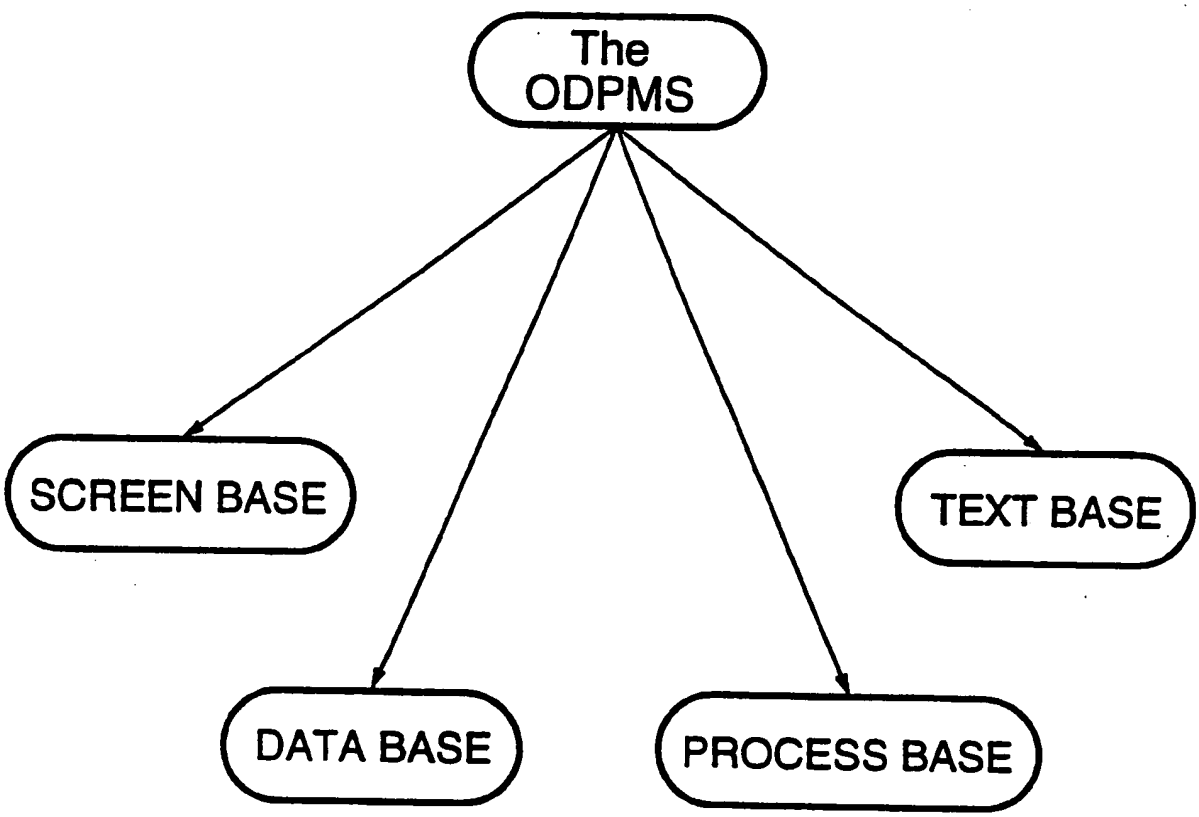
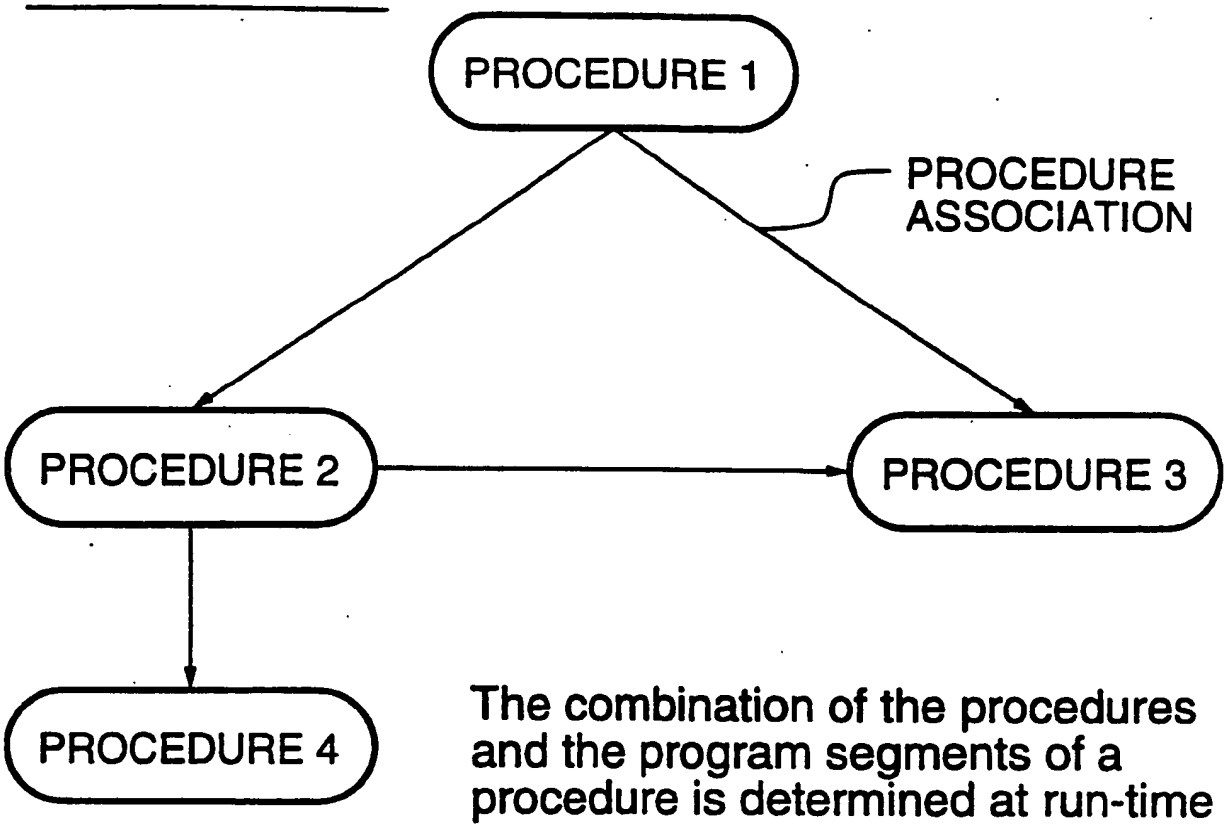


FIG.3.

A PROCESS BASE



The combination of the procedures and the program segments of a procedure is determined at run-time

The program segments of a procedure

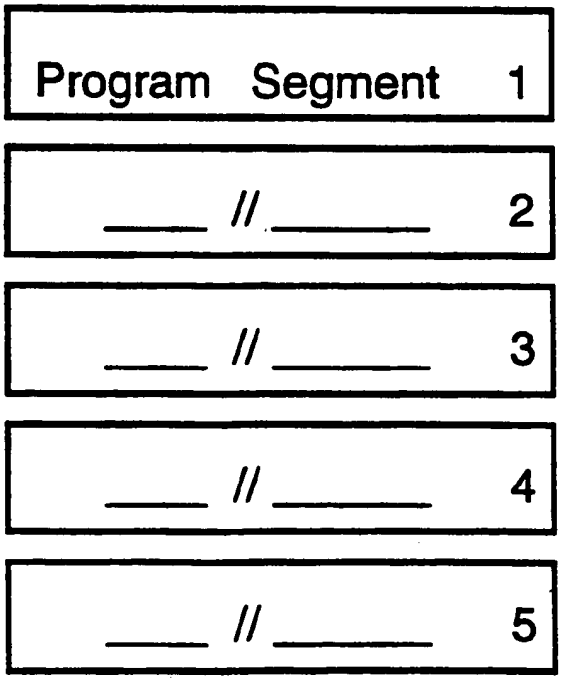
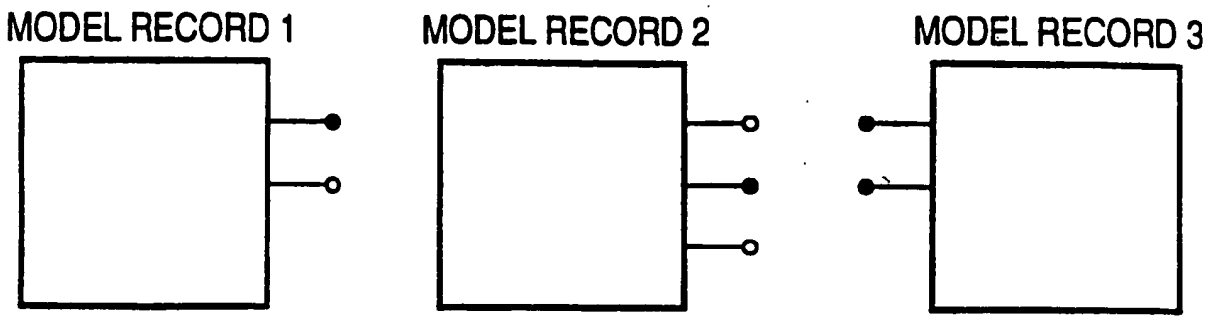


FIG.4.

THE ODPM



The symbol —● represents a fixed combinational interpretation and a fixed related object.

The symbol —○ represents a variable combinational interpretation and a variable related object.

The model segments of a model record

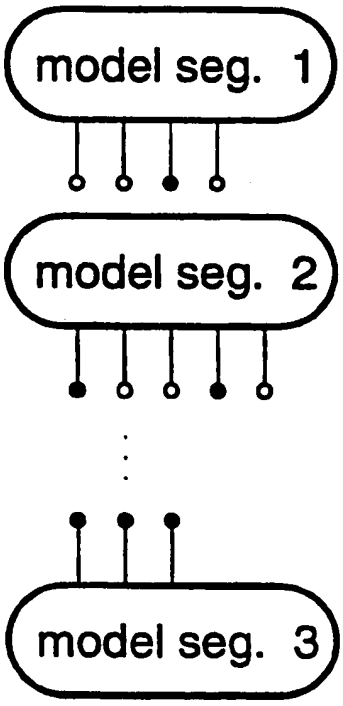


FIG.5.

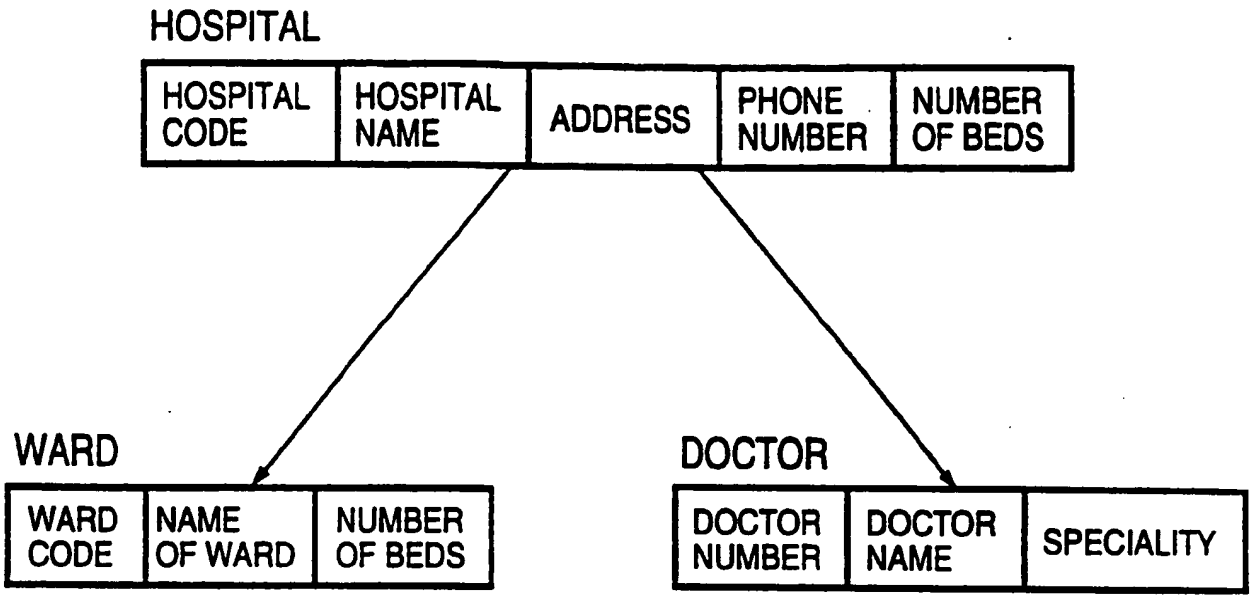
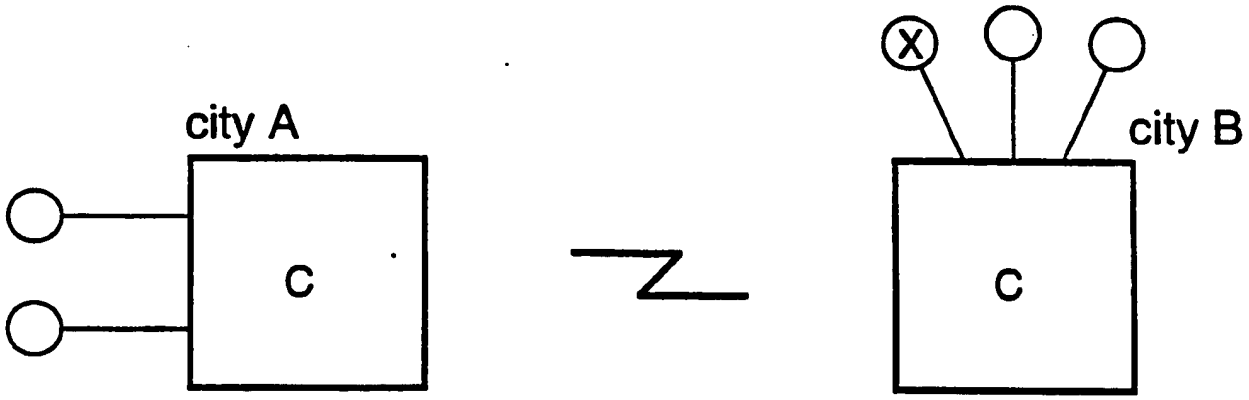


FIG.6.



1) A network system where C is any Mainframe, mini or PC.

2) X is the terminal being used

FIG.7.

THIS FUNCTION EXHIBITS
HOSPITAL INFORMATION

1) please enter the HOSPITAL CODE:

2) HOSPITAL NAME :

3) ADDRESS :

4) PHONE NUMBER :

5) NUMBER OF BEDS :

FIG.8A.

THIS FUNCTION EXHIBITS
DOCTOR'S INFORMATION

1) DOCTOR NUMBER :

2) DOCTOR NAME :

3) SPECIALITY :

FIG.8B.

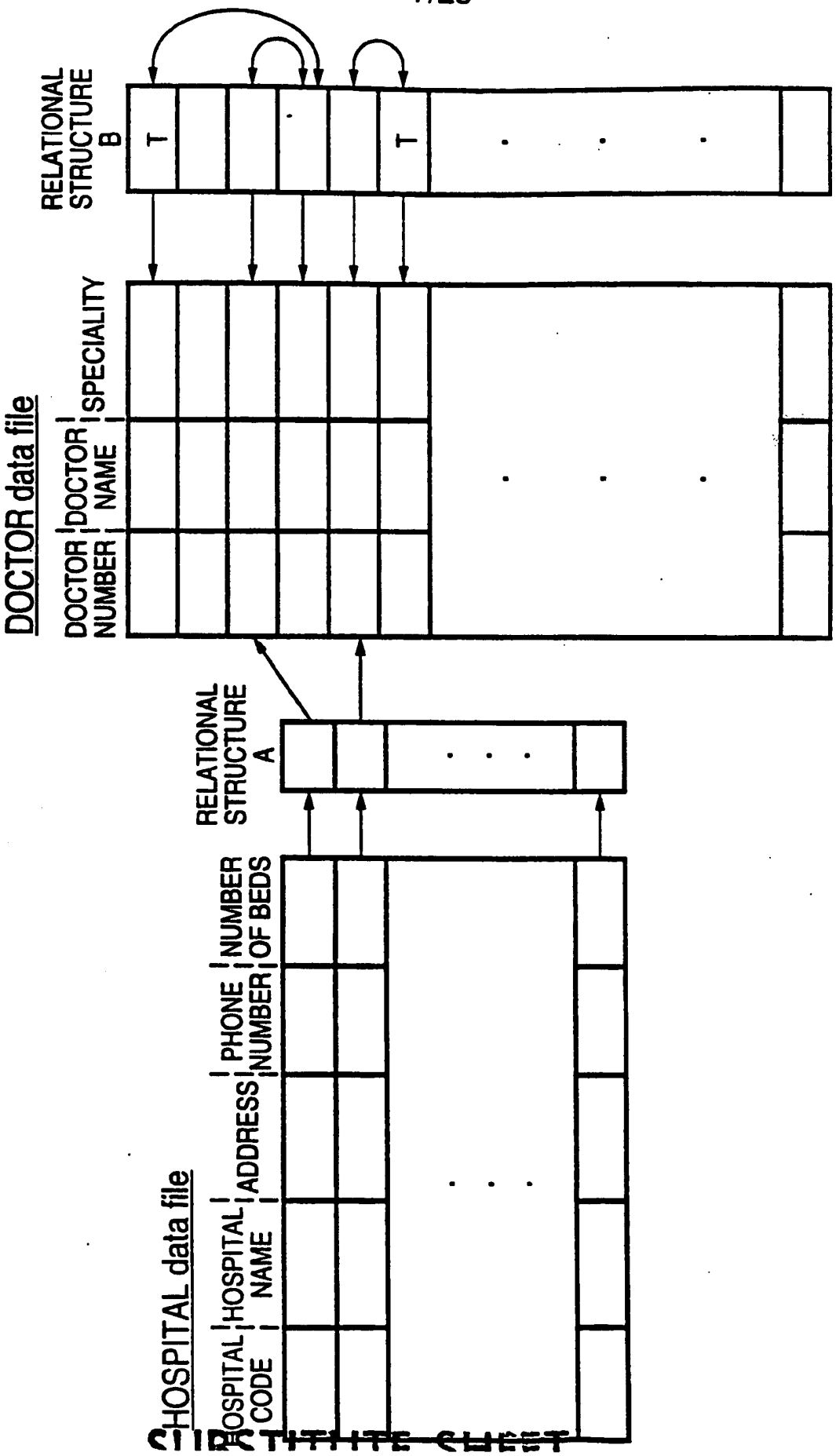


FIG.9.

HOSPITAL....INFO

YOU MUST HAVE ACCESS PERMISSION

To access any information in the data base
which exists in City A

You must first define HOSPITAL information.

First, type the name 'HOSPITAL'.

Then, use the "procedural relation":

'<screen....record....name>'

(i.e. <DOCTOR)

From HOSPITAL, you can access information
in the DOCTOR file by using
the "procedural relation":

'<DOCTOR1'

For the syntax and semantics of the symbols

of the ODPMS type the symbol ';'any place.

FIG.10.

```
int hospital (hospital_code, hospital_name, address, phone_number, number_of_beds)
        -Variable Definition-
{
 The ODPMS (hospital_code, hospital_name, address, phone_number, number_of_beds);
 interpretation of messages from the ODPMS;
 Programmer's damands (if any) to the source site;
}

int doctor (doctor_number, doctor_name, speciality)
        -Variable Definition-
{
 The ODPMS (doctor_number, doctor_name, speciality);
 interpretation of messages from the ODPMS;
 Programmer's demands (if any) to the source site;
}
```

FIG.11.

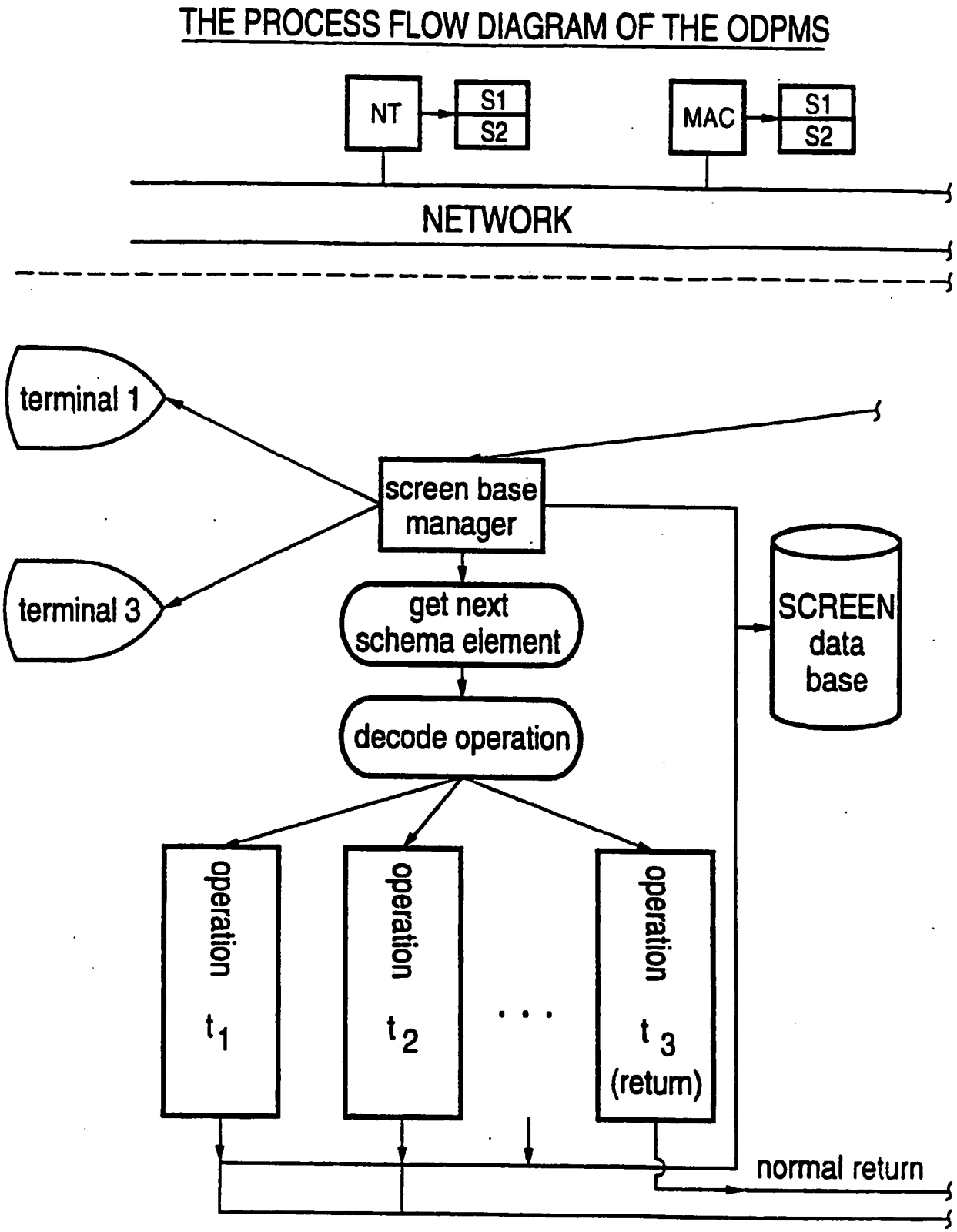

THE PROCESS FLOW DIAGRAM OF THE ODPMS
NT
S1
S2
MAC
S1
S2
NETWORK
terminal 1
terminal 3
screen base manager
get next schema element
decode operation
SCREEN data base
operation t1
operation t2
operation t3 (return)
normal return

FIG.12A.

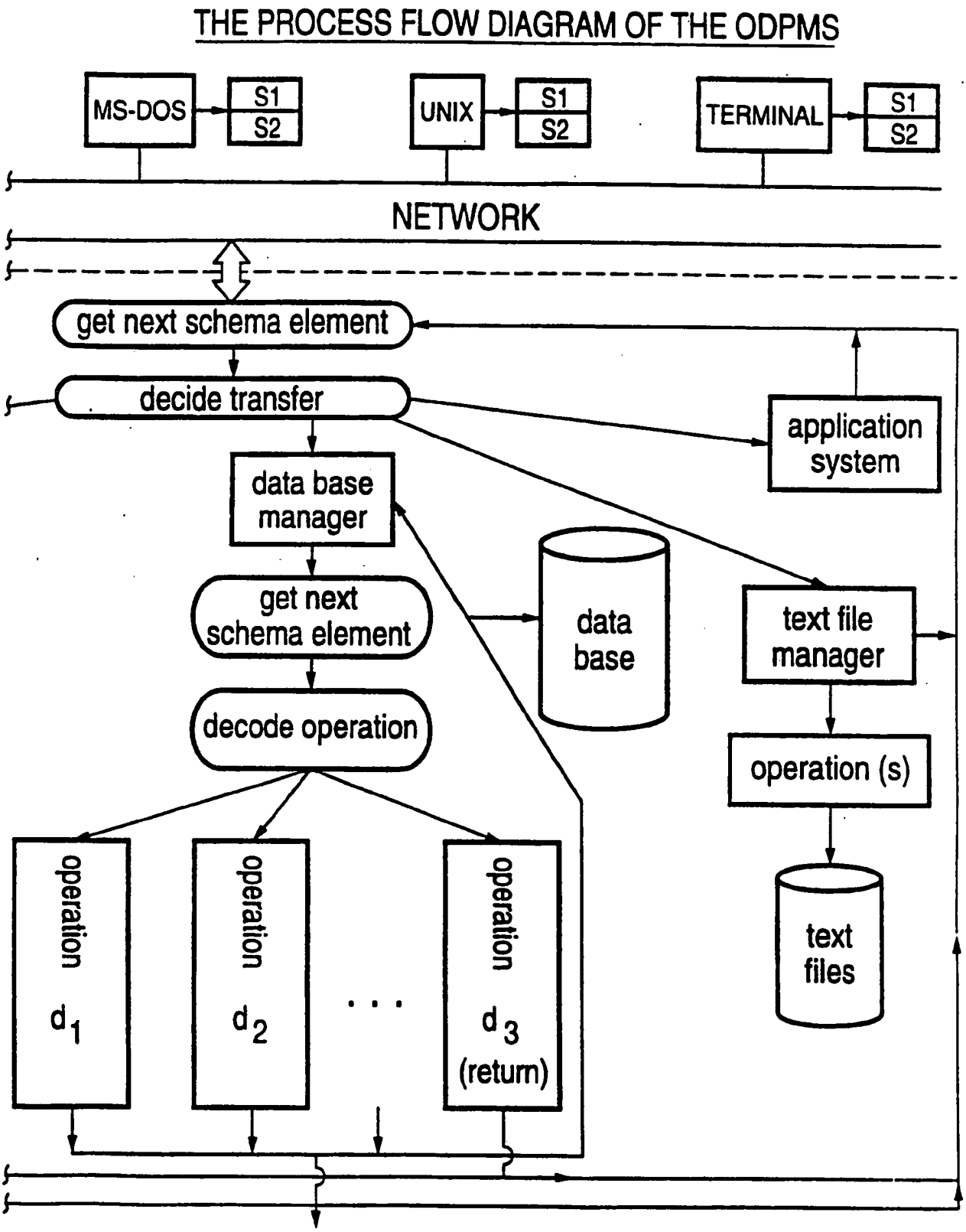
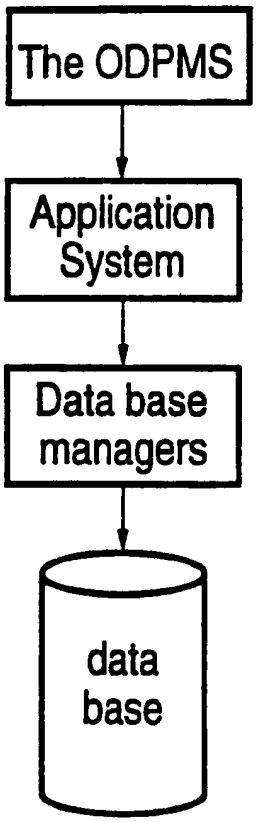


FIG.12B.

SUBSTITUTE SHEET

The Structures of S1 and S2 of Figure

The system S1 is using the ODPMS as a general data manager. Thus, the structure of S1 is:



The system S2 is using an other data base (s) manager(s). In this case, the structure of S2 is:

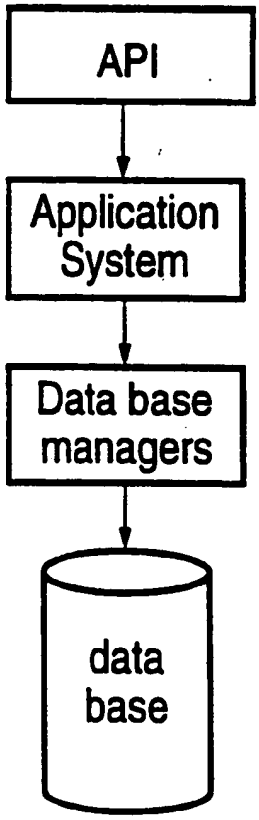


FIG.13.

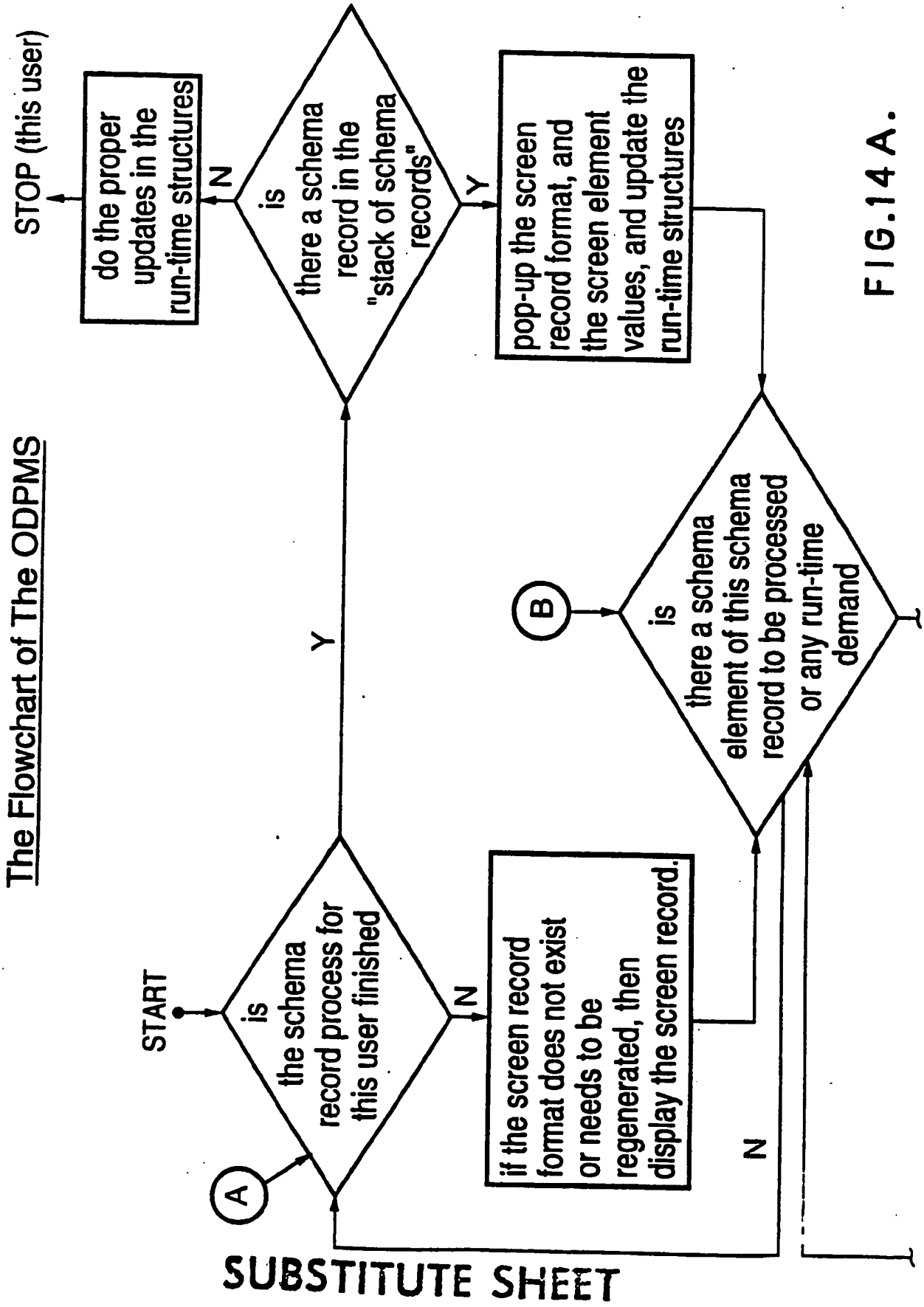

The Flowchart of The ODPMS
START
is the schema record process for this user finished
A
N
Y
if the screen record format does not exist or needs to be regenerated, then display the screen record.
B
is there a schema element of this schema record to be processed or any run-time demand
N
Y
is there a schema record in the "stack of schema records"
N
Y
do the proper updates in the run-time structures
pop-up the screen record format, and the screen element values, and update the run-time structures
STOP (this user)

FIG.14A.

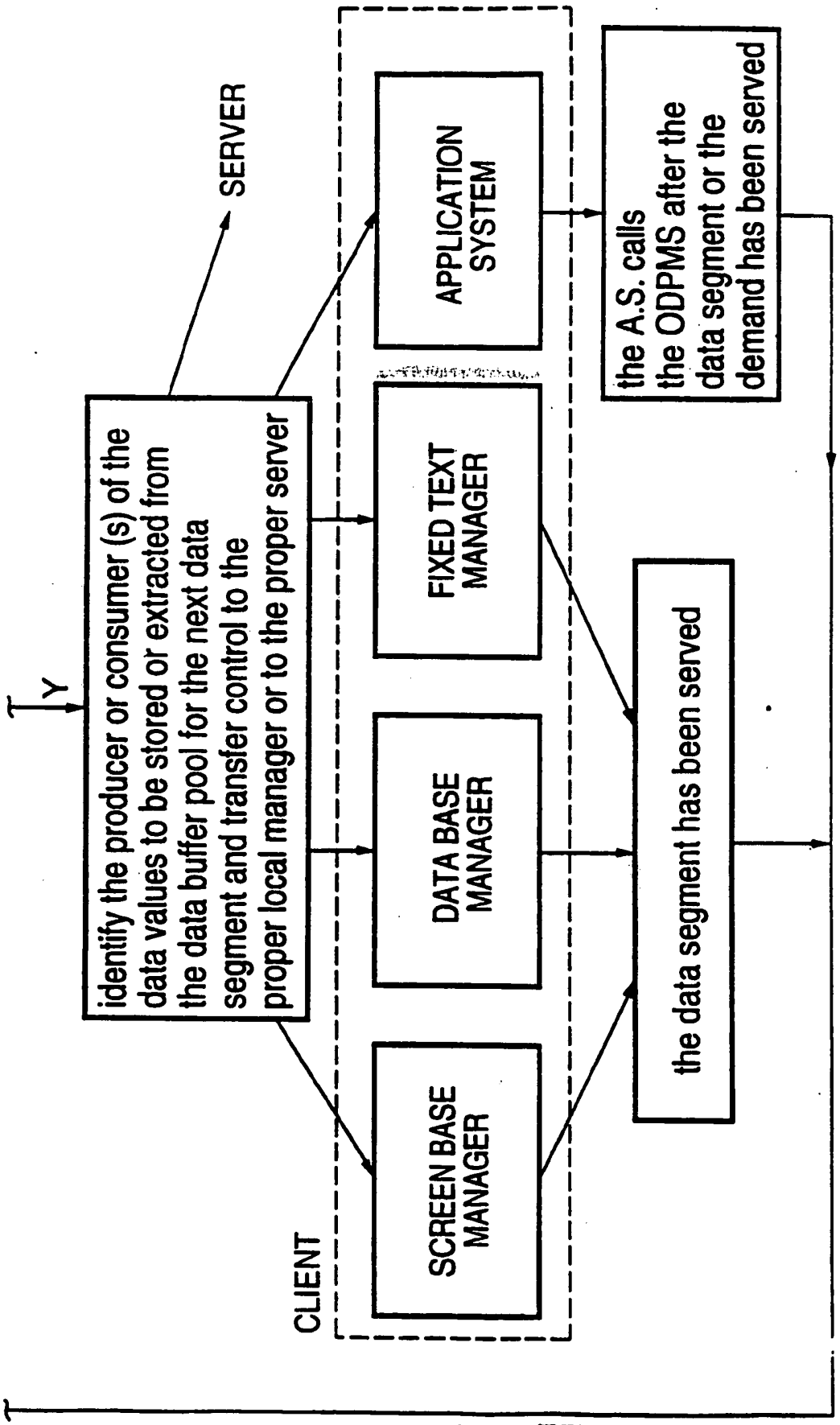

SERVER
identify the producer or consumer (s) of the data values to be stored or extracted from the data buffer pool for the next data segment and transfer control to the proper local manager or to the proper server
Y
CLIENT
APPLICATION SYSTEM
FIXED TEXT MANAGER
DATA BASE MANAGER
SCREEN BASE MANAGER
the A.S. calls the ODPMS after the data segment or the demand has been served
the data segment has been served

FIG.14B.

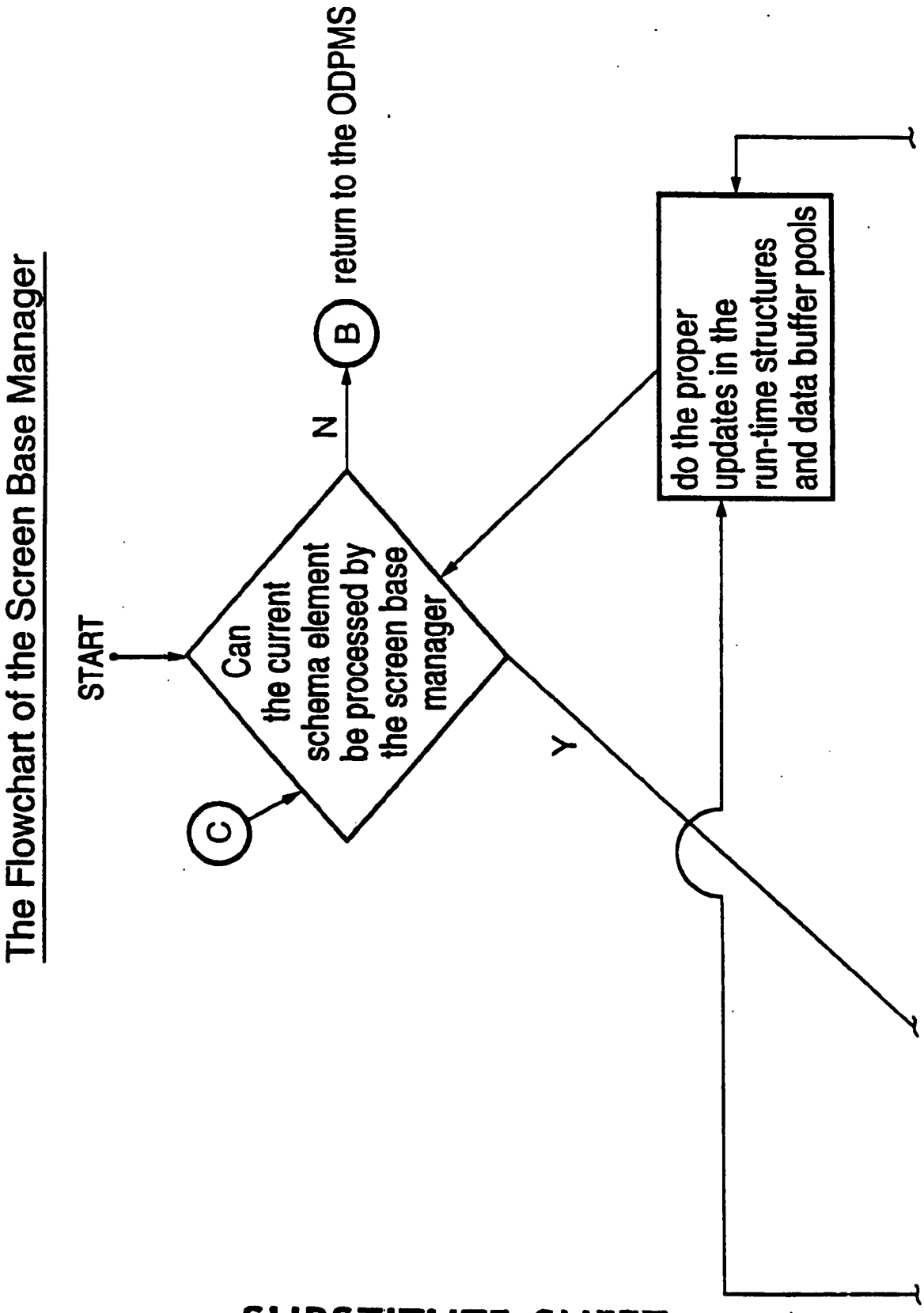

The Flowchart of the Screen Base Manager
START
Can the current schema element be processed by the screen base manager
C
N
B
return to the ODPMS
Y
do the proper updates in the run-time structures and data buffer pools

FIG.15A.

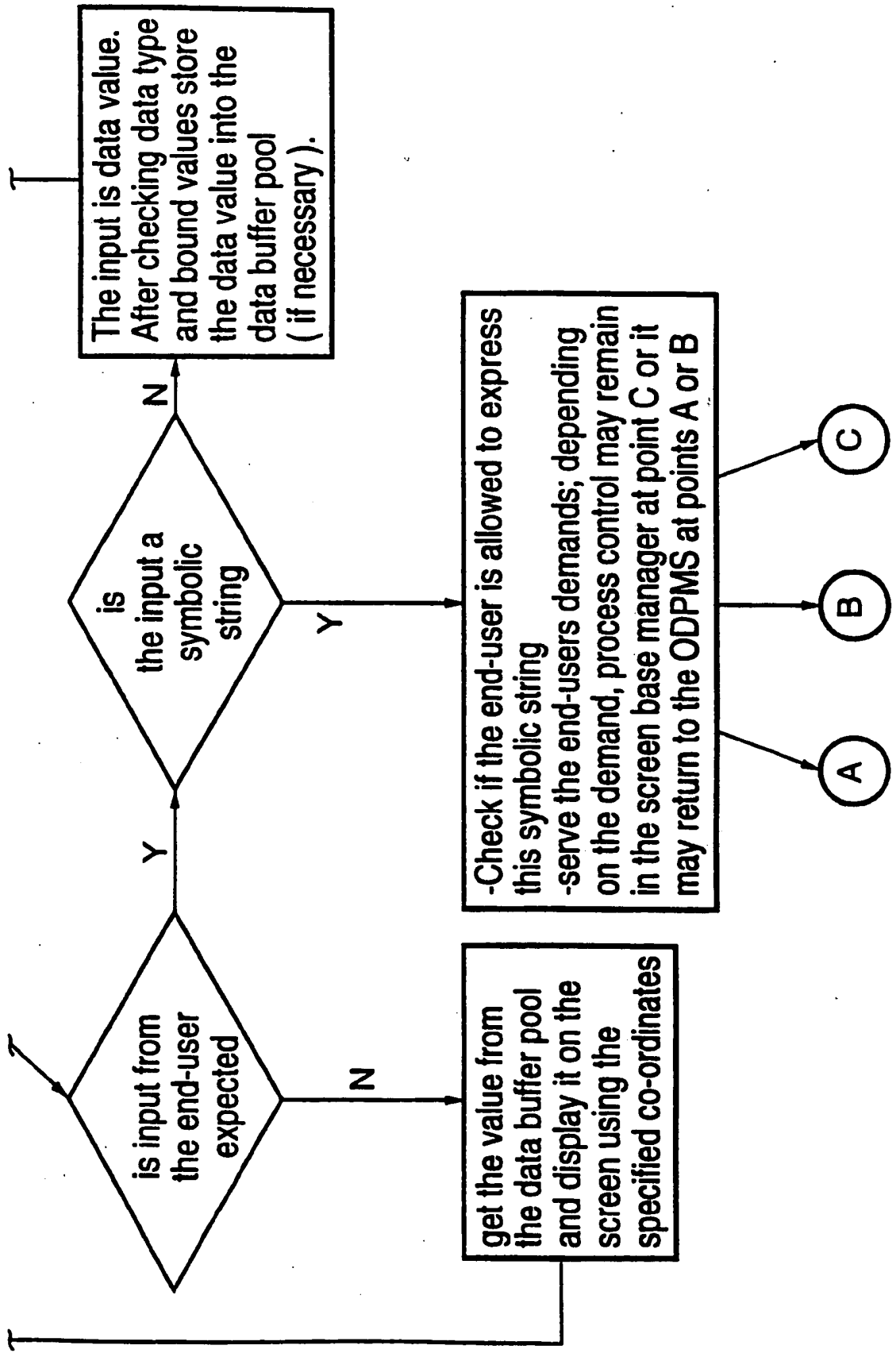

is input from the end-user expected
N
get the value from the data buffer pool and display it on the screen using the specified co-ordinates
Y
is the input a symbolic string
Y
-Check if the end-user is allowed to express this symbolic string
-serve the end-users demands; depending on the demand, process control may remain in the screen base manager at point C or it may return to the ODPMS at points A or B
A
B
C
N
The input is data value. After checking data type and bound values store the data value into the data buffer pool (if necessary).

FIG.15B.

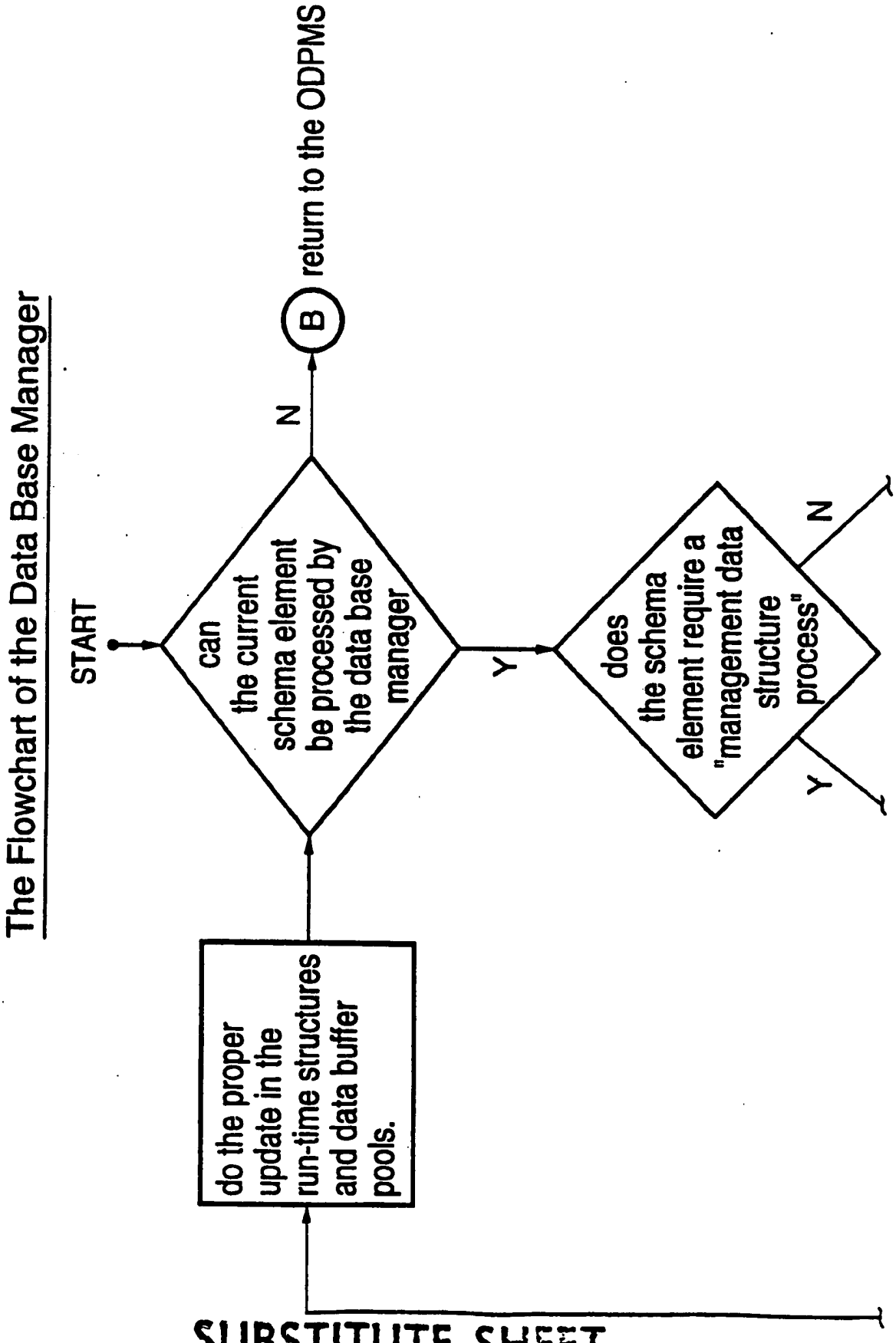
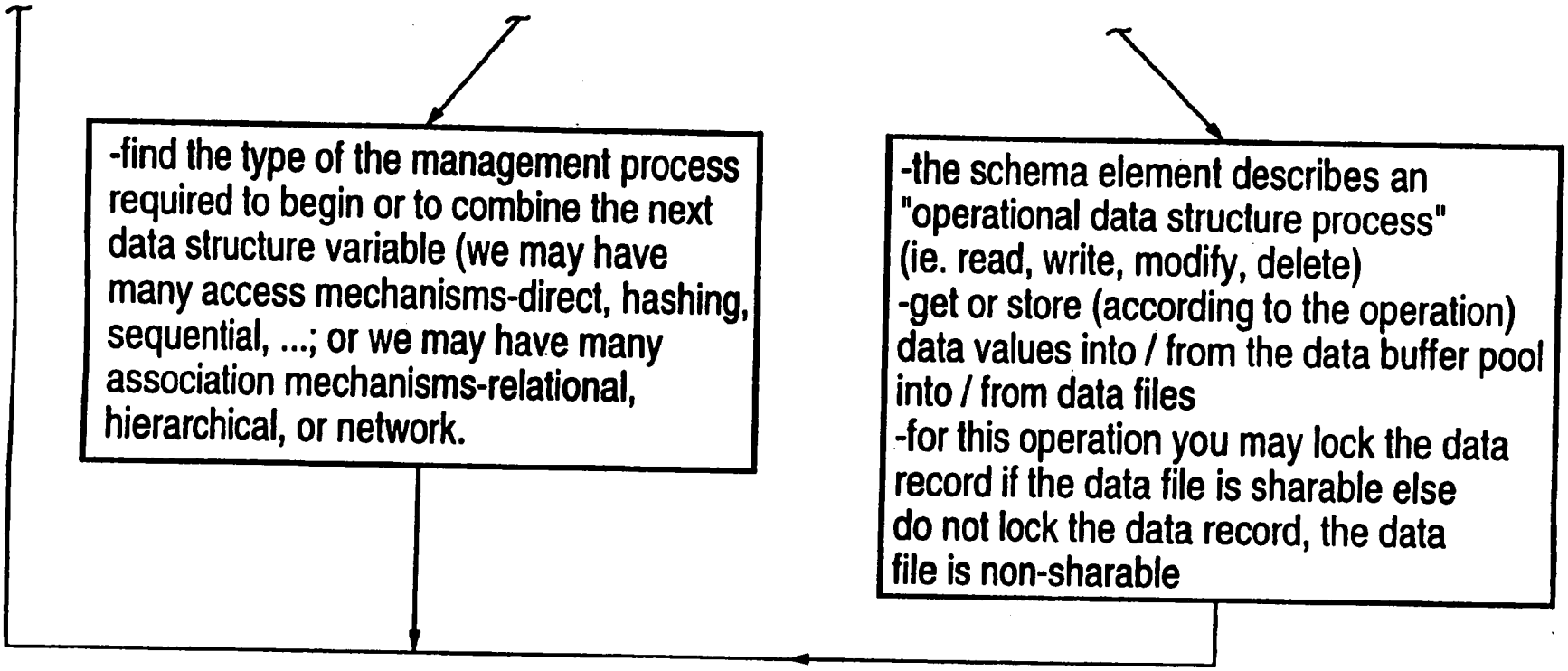

The Flowchart of the Data Base Manager
START
can the current schema element be processed by the data base manager
N
B
return to the ODPMS
Y
does the schema element require a "management data structure process"
N
Y
do the proper update in the run-time structures and data buffer pools.

FIG.16A.



Definitions:

A management data structure process: It starts a new logical data view or determines how a new logical data view is to be combined with the existing logical data view.

An operation data structure process: It determines the initial values or the changes of the data values of a data record.

FIG.16B.

The Flowchart of the Fixed Text Manager

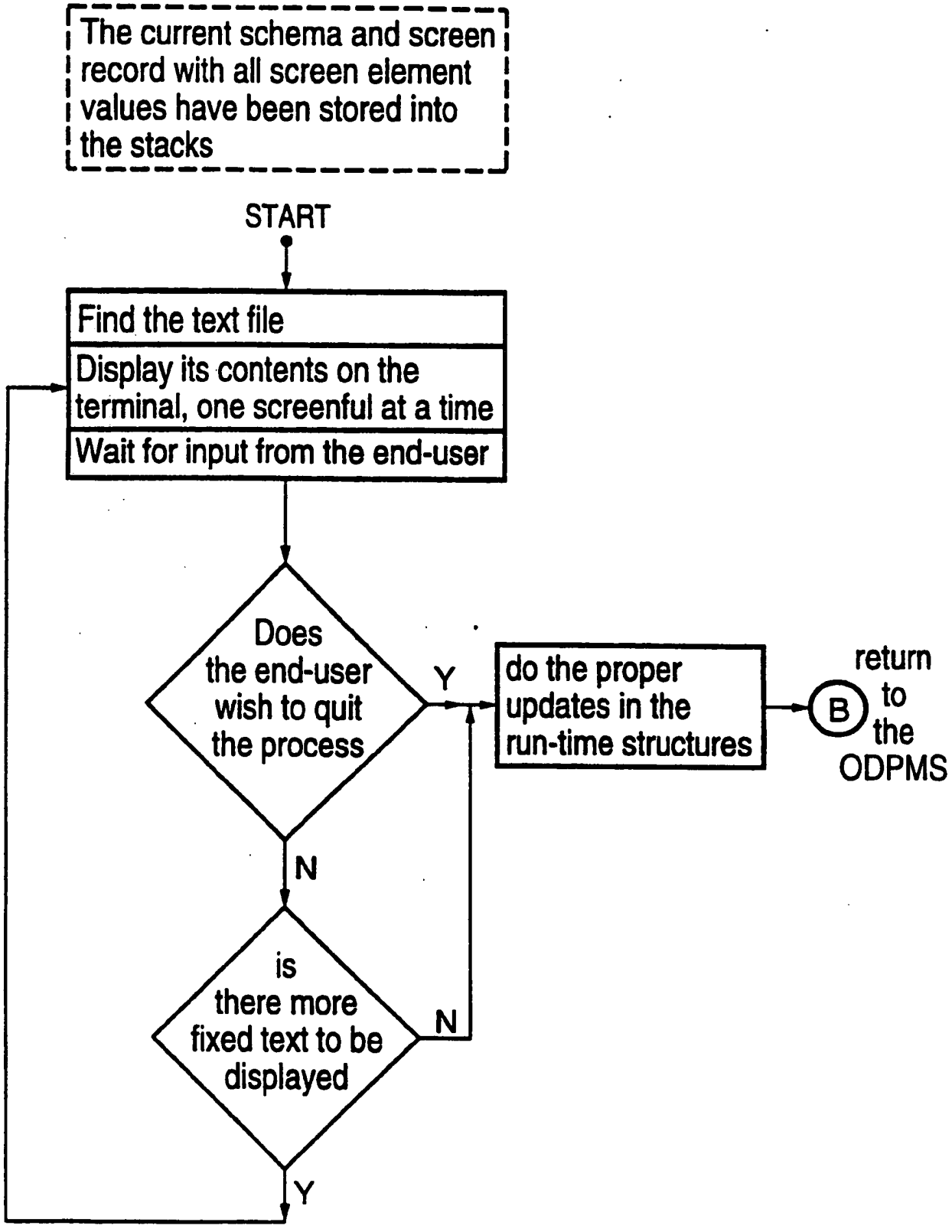


FIG. 17

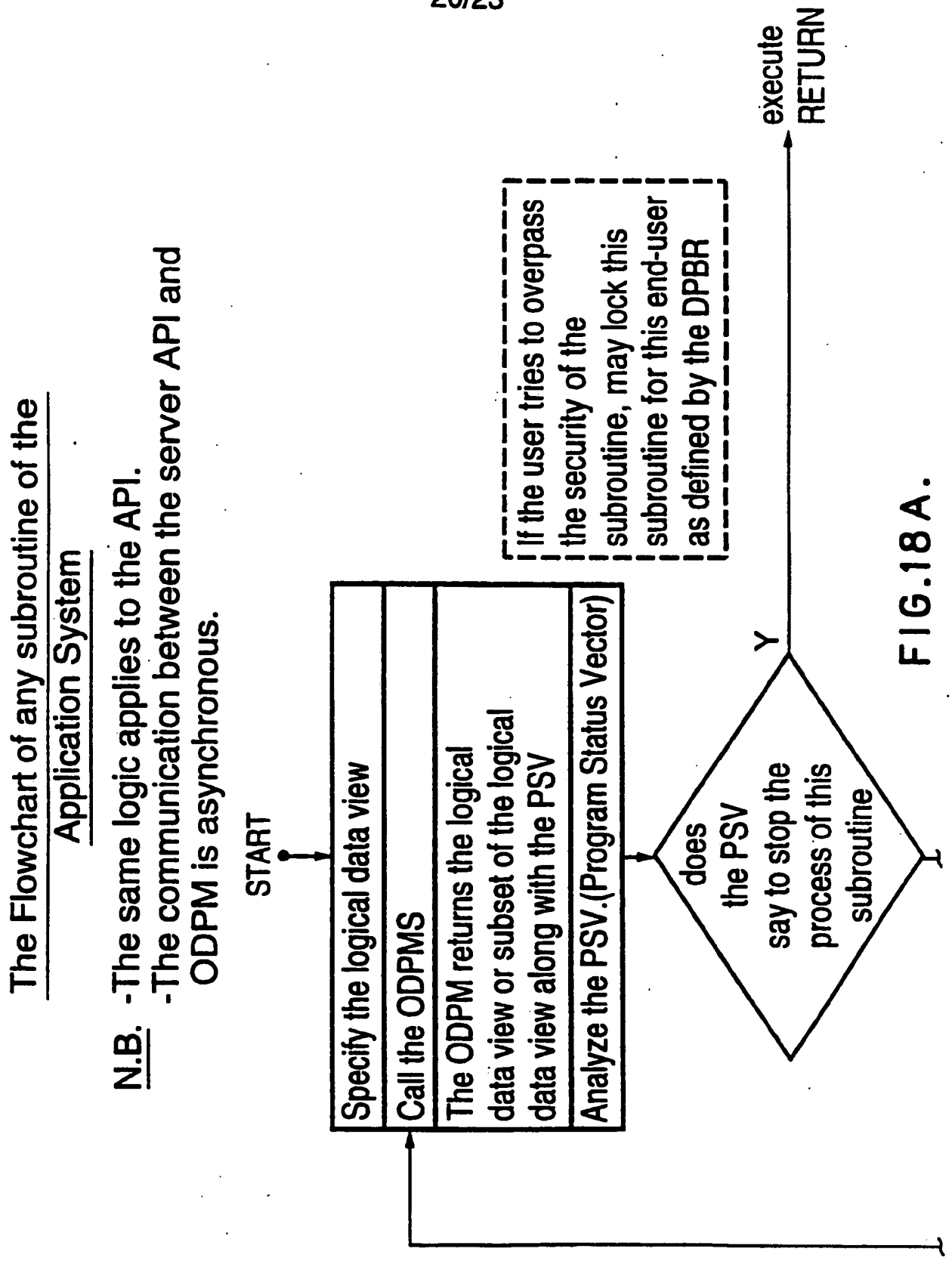

The Flowchart of any subroutine of the Application System
N.B. -The same logic applies to the API.
-The communication between the server API and ODPM is asynchronous.
START
Specify the logical data view
Call the ODPMS
The ODPM returns the logical data view or subset of the logical data view along with the PSV
Analyze the PSV.(Program Status Vector)
does the PSV say to stop the process of this subroutine
Y
execute RETURN
If the user tries to overpass the security of the subroutine, may lock this subroutine for this end-user as defined by the DPBR

FIG.18A.

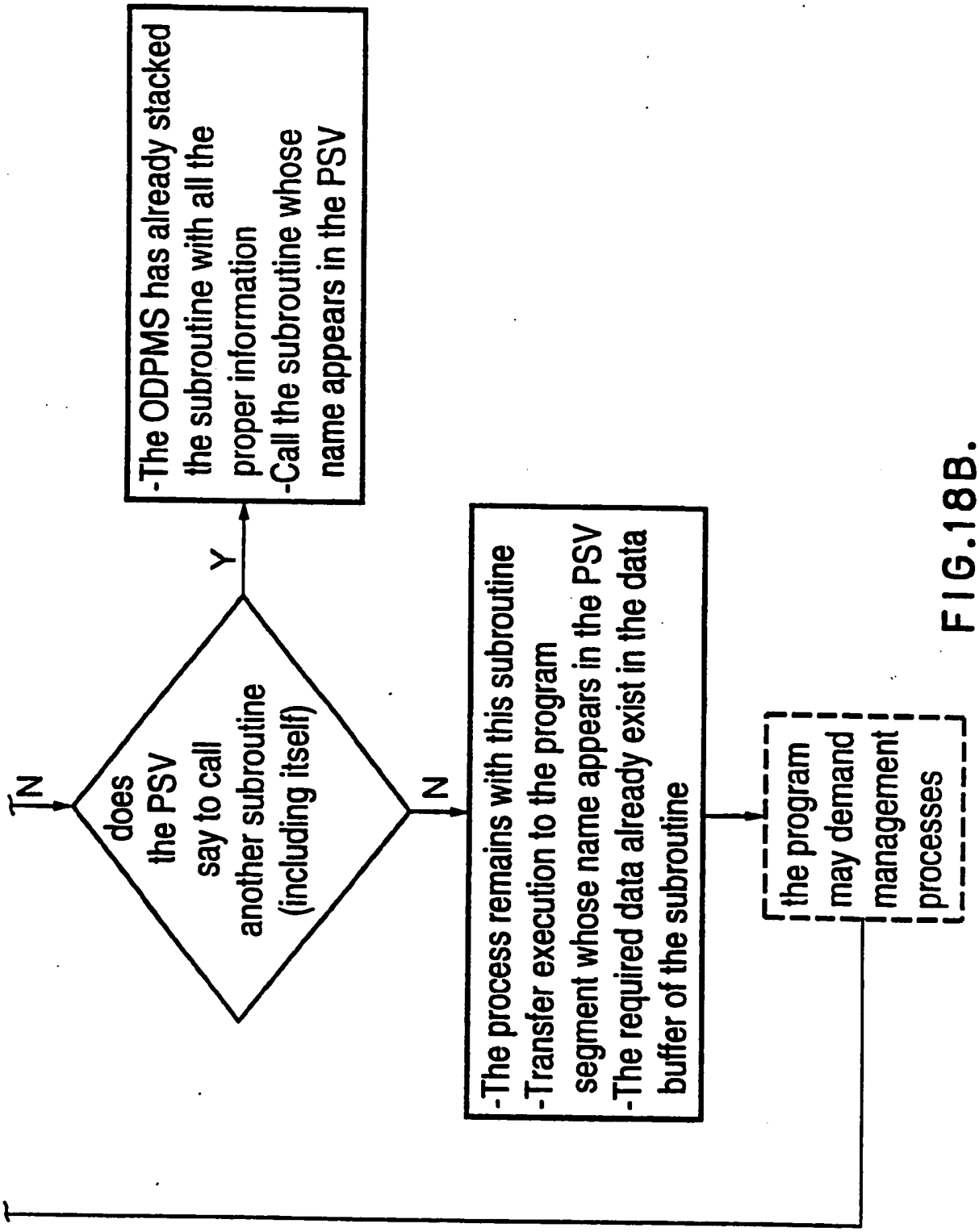

does the PSV say to call another subroutine (including itself)
Y
-The ODPMS has already stacked the subroutine with all the proper information
-Call the subroutine whose name appears in the PSV
N
N
-The process remains with this subroutine
-Transfer execution to the program segment whose name appears in the PSV
-The required data already exist in the data buffer of the subroutine
the program may demand management processes

FIG.18B.

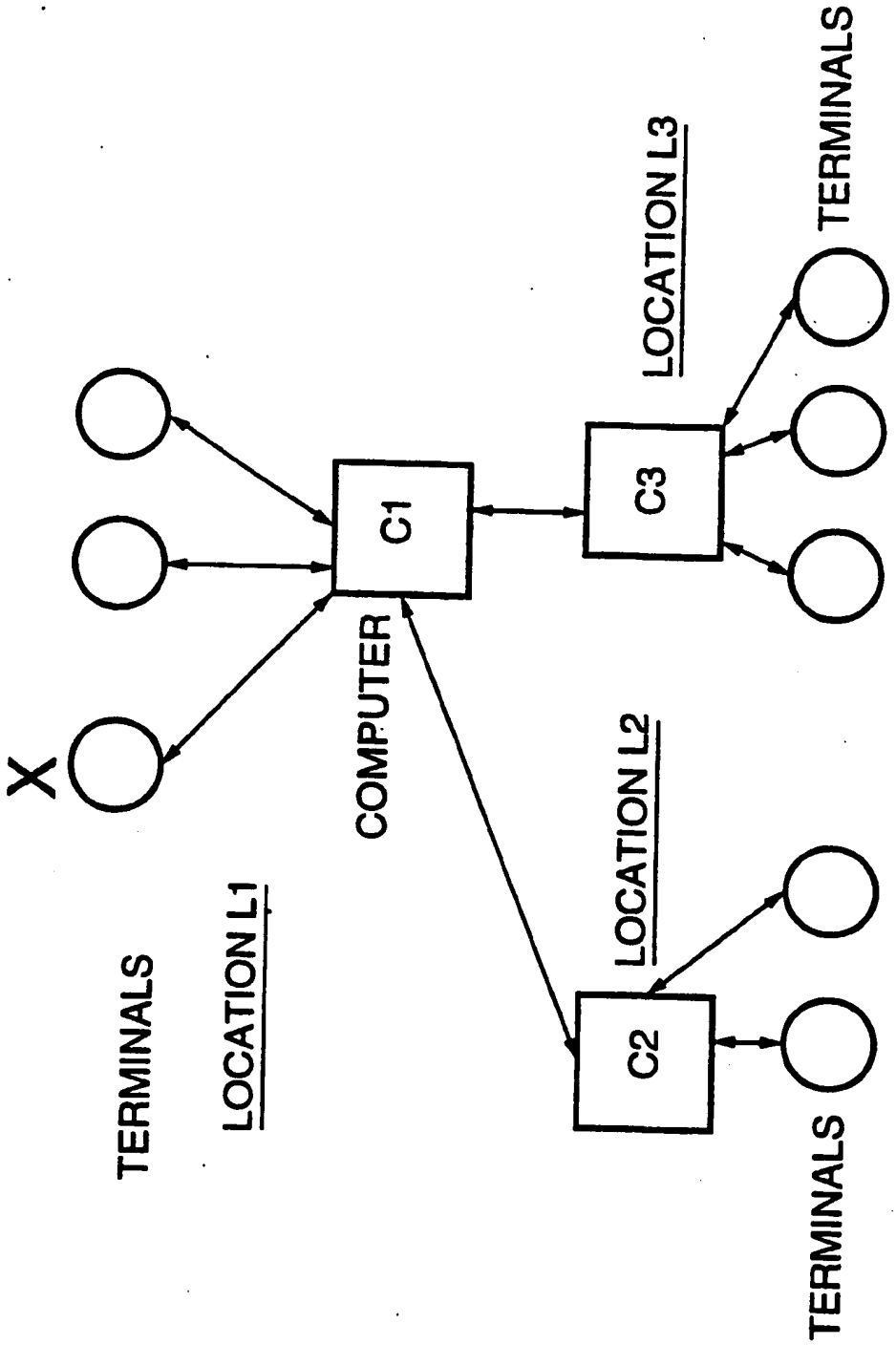

TERMINALS
LOCATION L1
X
COMPUTER
C1
C2
LOCATION L2
C3
LOCATION L3
TERMINALS
TERMINALS

FIG.19.

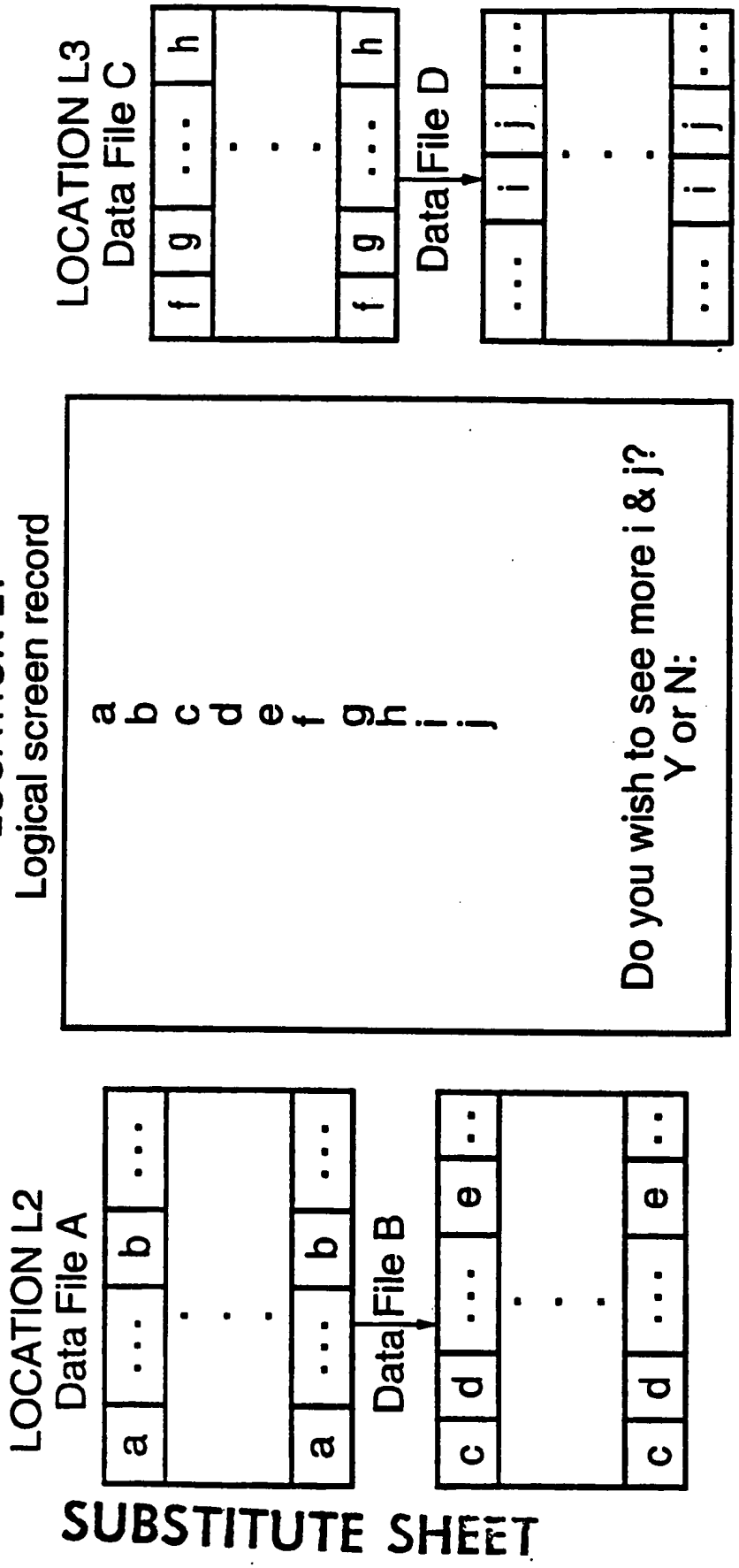

LOCATION L3
Data File C
f g ... h
f g ... h
Data File D
... i j ...
... i j ...
LOCATION L1
Logical screen record
a
b
c
d
e
f g
h
i j
Do you wish to see more i & j?
Y or N:
LOCATION L2
Data File A
a ... b ...
a ... b ...
Data File B
c d ... e ..
c d ... e ..

FIG.20.

International application No.
PCT/CA 94/00111

A. CLASSIFICATION OF SUBJECT MATTER
IPC 5 G06F15/403

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 5 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	PROCEEDINGS OF THE IEEE vol. 75, no. 5 , May 1987 , NEW YORK US pages 621 - 631 W. LITWIN ET AL 'An Overview of the Multi-Database Manipulation Language MDSL' see page 622, column 2, line 35 - page 623, column 1, line 42 ---	1-7,10, 12
X	COMPUTER vol. 24, no. 12 , December 1991 , LONG BEACH US pages 19 - 26 R. AHMED ET AL 'The Pegasus Heterogeneous Multidatabase System' see page 20, column 2, line 38 - page 21, column 3, line 15; figure 1	8,9,13
A	---	1-7, 10-12
	-/--	

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
12 July 1994	20. 07. 94

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax (+31-70) 340-3016	Fournier, C

International application No.
PCT/CA 94/00111

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	COMPUTER COMMUNICATIONS. vol. 15, no. 4 , May 1992 , GUILDFORD GB pages 270 - 278 M. KAMEL ET AL 'Federated database management system: Requirements, issues and solutions' see page 271, column 1, line 1 - page 273, column 1, line 11; figure 5	1-13
A	EP,A,0 474 340 (DEC) 11 March 1992 see column 1, line 17 - column 3, line 9	1-13

Patent document cited in search report	Publication date	Patent family member(s)		Publication date
EP-A-0474340	11-03-92	AU-B-	639802	05-08-93
		AU-A-	7945491	26-03-92
		CA-A-	2049121	15-02-92
		JP-A-	6075888	18-03-94